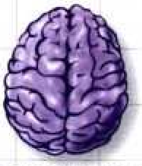
重视大脑的学习指南

Head First Python（中文版）





将重要的编程概念直接送入你的大脑



用列表、集合和字典对数据建模



将数据保存在pickle中



连接JSON、Android和App Engine



把你的定制应用移植到Web上



在PyPI上与全世界共享你的代码

Paul Barry 著
林琪 郭静 等译

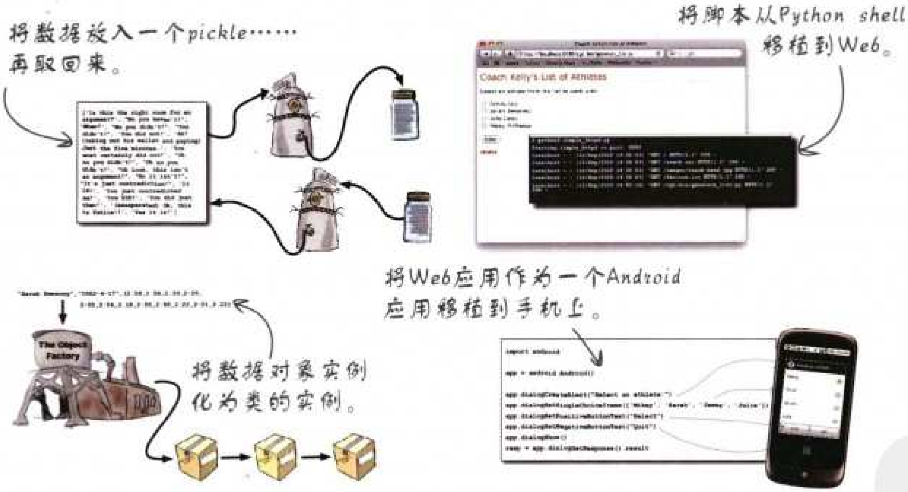
O'REILLY® 中国电力出版社

Head First Python（中文版）

Python/Programming Languages

您将从本书学会什么？

你想过可以通过一本书就学会Python吗？《Head First Python（中文版）》超越枯燥的语法和用法手册，通过一种独特的方法教你学习这种语言。你会迅速掌握Python的基础知识，然后转向持久存储、异常处理、Web开发、SQLite、数据加工和Google App Engine。你还将学习如何为Android编写移动应用，这都要归功于Python为你赋予的强大能力。本书会提供充分并且完备的学习体验，帮助你成为一名真正的Python程序员。



这本书为何与众不同？

我们觉得你的时间相当宝贵，不应当过多地花费在与新概念的纠缠之中。通过应用认知科学和学习理论的最新研究成果，《Head First Python（中文版）》可以让你投入一个需要多感官参与的学习体验，这本书采用丰富直观的形式使你的大脑真正开动起来，而不是长篇累牍地说教，让你昏昏欲睡。

"《Head First Python（中文版）》不单纯是一本优秀的Python语言入门书，更棒的是，它充分展示了Python在现实世界中如何使用。这本书并不是罗列干巴巴的语法，它会教你如何为Android手机、Google App Engine等创建应用程序。"

—— David Griffiths，
图书作者和敏捷教练

"其他书总是先从理论入手，然后过渡到示例，《Head First Python（中文版）》则不然，它直接进入代码，并随着内容的展开逐步对理论做出解释。书中提供的大量示例和解释足以涵盖你在日常工作中将要用到的大部分内容。"

—— Jeremy Jones，
《Python for Unix and Linux System Administration》
作者之一

O'REILLY®
oreilly.com.cn
headfirstlabs.com

O'Reilly Media, Inc.授权中国电力出版社出版

ISBN 978-7-5123-2223-3
9 787512 322233 >
定价：68.00元

此简体中文版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

Head First Python的作者



Paul Barry最近发现他的编程生涯已近四分之一个世纪，这个事实着实让人有些震惊。在此期间，Paul使用过多种不同的编程语言编写程序，他在两个大洲的两个国家生活并工作过，并且娶妻生子，如今已经有3个孩子（当然……实际上孩子们都是他妻子在悉心照顾，不过Paul确实在他们身边），另外他还攻读了计算机的学士和硕士学位，编写或合作编写了另外3本书，还为《Linux Journal》（他是这家杂志的特约编辑）撰写了大量技术文章。

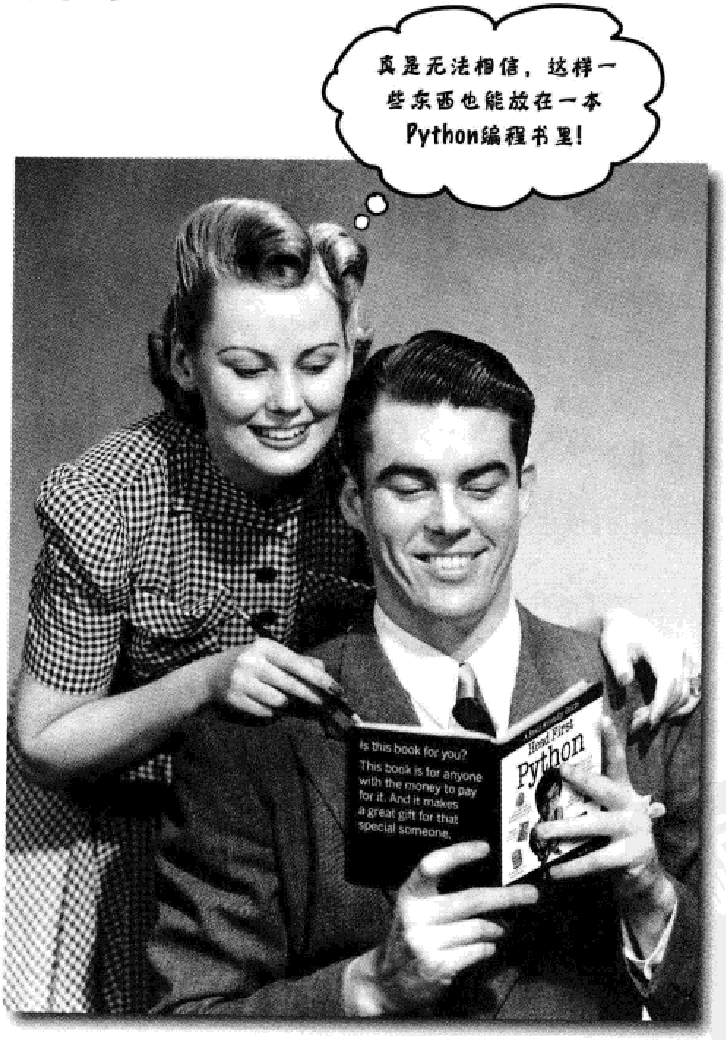
Paul从第一眼看到《Head First HTML with CSS & XHTML》就爱不释手，当时就意识到“Head First”方法必将成为教授编程的一种绝妙方法。那时他欣喜万分，同样兴奋的还有David Griffiths，他们共同完成了《Head First Programming》来证明当初的预感并非妄想。

Paul平日的工作是爱尔兰卡罗理工学院的一名讲师。作为计算与网络系的老师，Paul每天都在研究、学习以及向学生们传授编程技术，其中也包括Python。

最近Paul拿到了“课程与教学”研究生毕业证书，终于放心地发现他所做的大多数工作确实符合当今的第三级最佳实践。

如何使用这本书

引子



有一个问题真是听得我们耳朵都磨出茧了，这就是：“你们到底为什么要把这样一些东西放在一本Python书里呢？”这一部分正是要回答这个问题。

谁适合看这本书？

如果对下面的所有问题都能肯定地回答“是”：

1. 你是不是已经知道如何用另外一种编程语言编程？

2. 你是不是希望掌握Python编程的诀窍，想把它补充到你的工具集中，并用它完成一些新的创举？

3. 你是不是更愿意亲自动手，在实践中应用所学，而不只是听别人长篇大论地说教？

那么，这正是你要的书。

谁可能不适合看这本书？

如果满足下面任何一种情况：

1. 你是不是已经了解Python编程中需要知道的绝大多数内容？

2. 你是不是正在找一本Python参考书，希望它能极其详尽地涵盖所有细节？

3. 你是不是宁愿脚趾甲被15只尖叫的猴子拔掉也不愿意学新东西？是不是认为Python书就应该无所不包，即使这会让读者厌烦不已，也觉得这样反而更好？

那么，这本书并不适合你。



[来自市场的声音：任何一个有信用卡的人都可以拥有这本书……当然，我们也收支票。]

PDG

我们知道你在想什么

“这算是一本正式的Python书吗？”

“这些图用来做什么？”

“我真的能这样学吗？”



我们也知道你的大脑正在想什么

你的大脑总是渴求一些新奇的东西。它一直在搜寻、审视、期待着不寻常的事情发生。大脑的构造就是如此，正是这一点才让我们不至于墨守成规，能够与时俱进。

我们每天都会遇到许多按部就班的事情，这些事情很普通，对于这样一些例行的事情或者平常的东西，你的大脑又是怎么处理的呢？它的做法很简单，就是不让这些平常的东西妨碍大脑真正的工作。那么什么是大脑真正的工作呢？这就是记住那些确实重要的事情。它不会费心地去记乏味的东西。就好像大脑里有一个筛子，这个筛子会筛掉“显然不重要”的东西，如果遇到的事情枯燥乏味，这些东西就无法通过这个筛子。

那么你的大脑怎么知道到底哪些东西重要呢？打个比方，假如你某一天外出旅行，突然一只大老虎跳到你面前，此时此刻，你的大脑还有身体会做何反应？

神经元会“点火”，情绪爆发，释放出一些化学物质。

好了，这样你的大脑就会知道……

这肯定很重要！可不能忘记了！



不过，假如你正待在家里或者坐在图书馆里，这里很安全、很舒适，肯定没有老虎。你正在刻苦学习，准备应付考试。也可能想学一些比较难的技术，你的老板认为掌握这种技术需要一周时间，最多不超过十天。

这就存在一个问题。你的大脑很想给你帮忙。它会努力地把这些显然不太重要的内容赶走，保证这些东西不去侵占本不算充足的脑力资源。这些资源最好还是用来记住那些确实重要的事情，比如大老虎，遭遇火灾险情等。再比如，你的大脑会让你记住，绝对不能把“聚会”时狂欢的照片放在你的Facebook网页上。没有一种简单的办法来告诉大脑：“嘿，大脑，真是谢谢你了，不过不管这本书多没意思，也不管现在我对它多么无动于衷，但我确实希望你能把这些东西记下来。”

我们认为“Head First”读者就是要学习的人。

那么，怎么学习呢？首先必须获得知识，然后保证自己确实不会忘记。这可不是填鸭式的硬塞。根据认知科学、神经生物学和教育心理学的最新研究，学习的途径相当丰富，绝非只是通过书本上的文字。我们很清楚怎么让你的大脑兴奋起来。

下面是一些Head First学习原则：

看得到。与单纯的文字相比，图片更能让人记得住，通过图片，学习效率会更高（对于记忆和传递型的学习，甚至能有多达89%的效率提升）。而且图片更能让人看懂。以往总是把图片放在一页的最下面，甚至放在另外的一页上，与此不同，把文字放在与之相关的图片内部，或者在图片的周围写上相关文字，学习者的能力就能得到多至两倍的提高，从而能更好地解决有关问题。

采用一种针对个人的交谈式风格。最新的研究表明，如果学习过程中采用一种第一人称的交谈方式直接向读者讲述有关内容，而不是用一种干巴巴的语调介绍，学生在学习之后的考试中成绩会提高40%。正确的做法是讲故事，而不是做报告。要用通俗的语言。另外不要太严肃。如果你面对着这样两个人，一个是你在餐会上结识的很有意思的朋友，另一个人学究气十足，喋喋不休地对你说教，在这两个人中，你会更注意哪一个呢？

让学习的人想得更深。换句话说，除非你很积极地让神经元活动起来，否则你的头脑里什么也不会发生。必须引起读者的好奇，促进、要求并鼓励读者去解决问题、得出结论、产生新的知识。为此，需要发出挑战，留下练习题和拓宽思路的问题，并要求读者完成一些实践活动，让左右脑都开动起来，而且要利用到多种思维。

引起读者的注意，而且要让他一直保持注意。我们可能都有过这样的体验，“我真的想把这个学会，不过看过一页后实在是让我昏昏欲睡。”你的大脑注意的是那些不一般、有意思、有些奇怪、抢眼的、意料之外的东西。学习一项有难度的新技术并不一定枯燥。如果学习过程不乏味，你的大脑很快就能学会。

影响读者的情绪。现在我们知道了，记忆能力很大程度上取决于所记的内容对我们的情绪有怎样的影响。如果是你关心的东西，就肯定记得住。如果让你感受到了什么，这些东西就会留在你的脑海中。不过，我们所说的可不是什么关于男孩与狗的伤心故事。这里所说的情绪是惊讶、好奇、觉得有趣、想知道“什么……”还有就是一种自豪感，如果你解决了一个难题，学会了所有人都觉得很难的东西，或者发现你了解的一些知识竟是那些自以为无所不能的傲慢家伙所不知道的，此时就会有一种自豪感油然而生。

元认知：有关思考的思考

如果你真的想学，而且想学得更快、更深，就应该注意你怎样才会专注起来，考虑自己是怎样思考的，并了解你的学习方法。

我们中间大多数人长这么大可能都没有上过有关元认知或学习理论的课程。我们想学习，但是很少有人教我们怎么来学习。

不过,这里可以做一个假设，如果你手上有这本书，你想学习如何设计用户友好的网站，而且可能不想花太多时间。如果你想把这本书中读到的知识真正用起来，就需要记住你读到的所有内容。为此，必须理解这些内容。要想最大限度地利用这本书或其他任何一本书，或者掌握学习经验，就要让你的大脑负起责任，要求它记住这些内容。



怎么做到呢？技巧就在于要让你的大脑认为你学习的新东西确实很重要，对你的生活有很大影响。就像老虎出现在面前一样。如若不然，你将陷入旷日持久的拉锯战中，虽然你很想记住所学的新内容，但是你的大脑却会竭尽全力地把它们拒之门外。

那么究竟怎样才能让你的大脑把编程看作是一只饥饿的老虎呢？

这有两条路，一条比较慢，很乏味。另一条路不仅更快，还更有效。慢方法就是大量地重复。你肯定知道，如果反反复复地看到同一个东西，即便再没有意思，你也能学会并记住。如果做了足够的重复，你的大脑就会说，“尽管看上去这对他来说好像不重要，不过，既然他这样一而再、再而三地看同一个东西，所以我觉得这应该是重要的。”

更快的方法是尽一切可能让大脑活动起来，特别是开动大脑来完成不同类型的活动。如何做到这一点呢？上一页列出的学习原则正是一些主要的做法，而且经证实，它们确实有助于让你的大脑全力以赴。例如，研究表明，把文字放在所描述图片的中间（而不是放在这一页的别处，比如作为标题，或者放在正文中），这样会让你的大脑更多地考虑这些文字与图片之间有什么关系，而这就会让更多的神经元点火。让更多的神经元点火 ＝ 你的大脑更有可能认为这些内容值得关注，而且很可能需要记下来。

交谈式风格也很有帮助，当人们意识到自己在与“别人”交谈时，往往会更专心，这是因为他们总想跟上谈话的思路，并能做出适当的发言。让人惊奇的是，大脑并不关心“交谈”的对象究竟是谁，即使你只是与一本书“交谈”，它也不会在乎！另一方面，如果写作风格很正统、干巴巴的，你的大脑就会觉得，这就像坐在一群人中被动地听人作报告一样，很没意思，所以不必在意对方说的是什么，甚至可能打瞌睡。

不过，图片和交谈风格还只是开始而已，能做的还有很多……

PDG

我们是这么做的：

我们用了很多图，因为你的大脑更能接受看得见的东西，而不是纯文字。对你的大脑来说，一幅图抵千言。如果既有文字又有图片，我们会把文字放在图片当中，因为文字处在所描述的图片中间时，大脑的工作效率更高，倘若把这些描述文字作为标题，或者“湮没”在别处的大段文字中，就达不到这种效果了。

我们采用了重复手法，会用不同方式，采用不同类型的媒体，运用多种思维手段来介绍同一个东西，目的是让有关内容更有可能储存在你的大脑中，而且在大脑中多个区域都有容身之地。

我们会用你想不到的方式运用概念和图片，因为你的大脑喜欢新鲜玩艺。在提供图和思想时，至少会含着一些情绪因素，因为如果能产生情绪反应，你的大脑就会投入更大的注意。而这会让你感觉到这些东西更有可能要被记住，其实这种感觉可能只是很幽默，让人奇怪或者比较感兴趣而已。

我们采用了一种针对个人的交谈式风格，因为当你的大脑认为你在参与一个会谈，而不是被动地听一场演示汇报时，它就会更加关注。即使你实际上在读一本书，也就是说在与书“交谈”，而不是真正与人交谈，但这对你的大脑来说并没有什么分别。

在这本书里，我们加入了80多个实践活动，因为与单纯的阅读相比，如果能实际做点什么，你的大脑会更乐于学习，更愿意去记。这些练习都是我们精心设计的，有一定的难度，但是确实能做出来，因为这是大多数人所希望的。

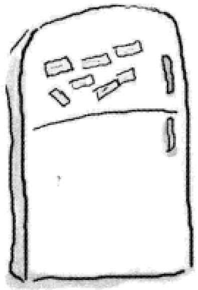
我们采用了多种学习模式，因为尽管你可能想循序渐进地学习，但是其他人可能希望先对整体有一个全面的认识，另外可能还有人只是想看一个例子。不过，不管你想怎么学，要是同样的内容能以多种方式来表述，这对每一个人都会有好处。

这里的内容不只是单单涉及左脑，也不只是让右脑有所动作，我们会让你的左右脑都开动起来，因为你的大脑参与得越多，你就越有可能学会并记住，而且能更长时间地保持注意力。如果只有一半大脑在工作，通常意味着另一半有机会休息，这样你就能更有效率地学习更长时间。

我们会讲故事，留练习，从多种不同的角度来看同一个问题，这是因为，如果要求大脑做一些评价和判断，它就能更深入地学习。

我们会给出一些练习，还会问一些问题，这些问题往往没有直截了当的答案，通过克服这些挑战，你就能学得更好，因为让大脑真正做点什么的话，它就更能学会并记住。想想吧，如果只是在体育馆里看着别人流汗，这对于保持你自己的体形肯定不会有什么帮助，正所谓临渊羡鱼，不如退而结网。不过另一方面，我们会竭尽所能不让你钻牛角尖，把劲用错了地方，而是能把功夫用在点子上。也就是说，你不会为搞定一个难懂的例子而耽搁，也不会花太多时间去弄明白一段艰涩难懂而且通篇行话的文字，我们的描述也不会太过简洁而让人无从下手。

我们用了拟人手法。在故事中，在例子中，还有在图中，你都会看到人的出现，这是因为你本身是一个人，不错，这就是原因。如果和人打交道，相对于某件东西而言，你的大脑会更为关注。



可以用下面的方法让你的大脑就范

好了，我们该做的已经做了，剩下的就要看你自己的了。以下提示可以作为一个起点：听一听你的大脑是怎么说的，弄清楚对你来说哪些做法可行，哪些做法不能奏效。要尝试新鲜事物。

把这一页撕下来，贴到你的冰箱上。

---

1 慢一点。你理解的越多，需要记的就越少。

不要光是看看就行了。停下来，好好想一想。书中提出问题的时候，你不要直接去翻答案。可以假想真的有人在问你这个问题。你让大脑想得越深入，就越有可能学会并记住它。

2 做练习，自己记笔记。

我们留了练习，但是如果这些练习的解答也由我们一手包办，那和有人替你参加考试有什么分别？不要只是坐在那里看着练习发呆。拿出笔来，写一写画一画。大量研究都证实，学习过程中如果能实际动动手，这将改善你的学习。

3 阅读“没有傻问题”。

顾名思义。这些问题不是可有可无的旁注，它们绝对是核心内容的一部分！千万不要跳过去不看。

4 上床睡觉之前不要再看别的书，至少不要看其他有难度的东西。

学习中有一部分是在你合上书之后完成的（特别是，要把学到的知识长久地记住，这往往无法在看书的过程中做到）。你的大脑也需要有自己的时间，这样才能再做一些处理。如果在这段处理时间内你又往大脑里灌输了新的知识，那么你刚才学的一些东西就会丢掉。

5 讲出来，而且要大声讲出来。

说话可以刺激大脑的另一部分。如果你想看懂什么，或者想更牢地记住它，就要大声地说出来。更好的办法是，大声地解释给别人听。这样你会学得更快，而且可能会有以前光看不说时不曾有的新发现。

6 要喝水，而且要大量喝水。

能提供充足的液体，你的大脑才能有最佳表现。如果缺水（可能在你感觉到口渴之前就已经缺水了），学习能力就会下降。

7 听听你的大脑怎么说。

注意一下你的大脑是不是负荷太重了。如果发现自己开始浮光掠影地翻看，或者刚看的东西就忘记了，这说明你该休息一会了。达到某个临界点时，如果还是一味地向大脑里塞，这对于加快学习速度根本没有帮助，甚至还可能影响正常的学习进程。

8 要有点感觉。

你的大脑需要知道这是很重要的东西。要真正融入到书中的故事里。为书里的照片加上你自己的图题。你可能觉得一个笑话很蹩脚，但这总比根本无动于衷要好。

9 编写大量软件！

要学习编程，没有别的办法，只能通过编写大量代码。这本书正是要这么做。编写代码是一种技巧，要想在这方面擅长，只能通过实践。我们会给你提供大量实践的机会：每一章都留有练习，提出问题让你解决。不要跳过这些练习，很多知识都是在完成这些练习的过程中学到的。我们为每个练习都提供了答案，如果你实在做不出来（很容易被一些小问题卡住），看看答案也无妨！不过在看答案之前，还是要尽力先自己解决问题。而且在读下一部分之前，一定要确确实实地掌握前面的内容。

重要说明

要把这看做是一个学习过程，而不要简单地把它看成是一本参考书。我们在安排内容的时候有意做了一些删减，只要是对有关内容的学习有妨碍，我们都毫不留情地把这些部分删掉。另外，第一次看这本书时，要从第一页从头看起，因为书中后面的部分会假定你已经看过而且学会了前面的内容。

这本书特别设计为使你能对Python尽快上手。

既然你想学真功夫，这里就会教你真功夫。所以，在这本书中不会看到长篇大论的技术内容，这里不会用干巴巴的表格罗列Python的操作符，也不会给出枯燥的操作符优先级规则。所有这些都没有，不过我们会精心安排，尽可能涵盖所有基础知识，使你能把Python尽快记入大脑并永远留住。我们只做了一个假设，认为你已经知道如何用另外某种编程语言编写程序。

这本书面向Python 3。

这本书中使用Python编程语言的版本3，第1章会介绍如何得到和安装Python 3。当然，我们并不是完全忽略版本2，这一点你在第8章到第11章就会发现。不过请相信，你会庆幸使用Python，因为你根本不会注意到你编程实现的技术是在Python 2上运行。

我们会直接让Python投入工作。

从第1章开始你就会用Python做些有用的工作。这里不会绕弯子，因为我们希望你能立即用Python开展工作。

书里的实践活动不是可有可无的。

这里的练习和实践活动不是可有可无的装饰和摆设，它们也是这本书核心内容的一部分。其中有些练习和活动有助于记忆，有些则能够帮助你理解，还有一些对于如何应用所学的知识很有帮助。千万不要跳过这些练习不做。

PDG

我们有意安排了许多重复，这些重复非常重要。

Head First系列图书有一个与众不同的地方，这就是，我们希望你确确实实地掌握这些知识，另外希望在学完这本书之后你能记住学过了什么。大多数参考书都不太重视重复和回顾，但是由于这是一本有关学习的书，你会看到一些概念一而再、再而三地出现。

代码例子尽可能短小精悍。

有读者告诉我们，如果查了200行代码才能找到要理解的那两行代码，这很让人郁闷。这本书里大多数例子往往都开门见山，作为上下文的代码会尽可能的少，这样你就能一目了然地看到哪些东西是需要你学习的。别指望这些代码很健壮，甚至别指望它们是完整的。我们特意把这些例子写得很简单，以便于你学习，它们的功能往往不太完备。

我们在网上放了大量代码示例，你可以根据需要复制和粘贴。可以从以下两个网址下载：

http://www.headfirstlabs.com/books/hfpython/

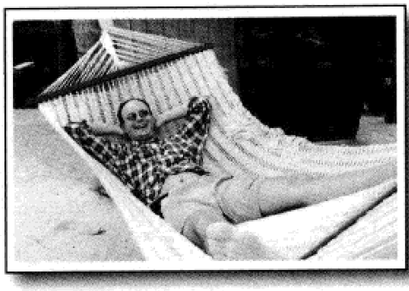
http://python.itcarlow.ie

“头脑风暴”练习没有答案。

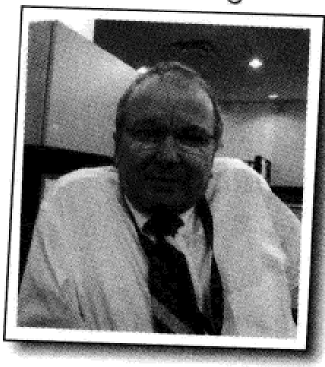
有一些头脑风暴练习根本没有正确的答案，对于另外一些练习，头脑风暴实践活动中有一部分学习过程就是让你确定你的答案是否正确以及在何种情况下正确。在其中一些头脑风暴练习中，你会得到一些提示来指出正确的方向。

技术审校团队

David Griffiths



Phil Hartley



Jeremy Jones



技术审校：

David Griffiths是《Head First Rails》的作者，同时也是《Head First Programming》的作者之一。他12岁时看到一份关于Seymour Papert工作的资料，从那时起就开始了他的编程生涯。15岁时，他编写了Papert的计算机语言LOGO的实现程序。在大学学习了纯粹数学之后，他开始为计算机编写代码，另外还为人们撰写杂志文章。他身兼多职，既是敏捷方法教练和开发人员，同时还是一个车厂修理工（不过这些角色并没有先后顺序）。他可以用十余种语言编写代码，也可以用一种语言完全搞定。如果不是在写书、编程或者辅导别人，他的闲暇时光大多会用来与他挚爱的妻子Dawn（也是Head First系列的作者之一）四处旅游。

Phil Hartley已获得苏格兰爱丁堡大学的计算机科学学位。经过在IT行业30多年的摸爬滚打，特别在OOP方面积累了丰富的经验。目前他在亚利桑那州滕比的高级科技大学任全职教师。空闲时间里，Phil是一个狂热的NFL球迷。

Jeremy Jones是《Python for Unix and Linux System Administration》的作者之一。2001年以来，他一直在积极地使用Python。他做过开发人员、系统管理员、质量工程师，还做过系统分析员。这些职业都各有回报和挑战，不过对他来说，最富有挑战同时回报最大的“职业”当属丈夫和父亲。

致谢

致我的编辑：

Brian Sawyer是这本书的编辑。如果不是在编辑图书，Brian空闲时喜欢跑马拉松。与我合作再出一本书（这是我们的二次联手）绝对是一个很好的训练。O'Reilly和Head First团队真的很幸运，能拥有像Brian这样有着过人才干的高手，使这本书以及其他书能够最完美地呈现在大家面前。



Brian Sawyer

致O'Reilly团队：

Karen Shaner提供了管理支持，并且运用他的卓越才能很好地协调了技术审校过程，对于我提出的众多请求和询问，都迅速作答，并全力提供帮助。还要感谢那些幕后的人们——O'Reilly制作团队，是他们指导这本书顺利进入最终的出版环节，将我的InDesign原稿文件变成你手上这本精美的图书（也可能你在用iPad或Android手机阅读，或者在你的PC机上读这本书，那同样也很酷）。

还要感谢Head First系列的其他作者，在本书的整个编写过程中，他们通过Twitter给予了我充分的赞赏、建议和鼓励。你可能想不到140个字符能带来天壤之别，但事实上确实如此。

另外还要向Bert Bates以及Kathy Sierra致以诚挚的谢意，正是从他们绝妙的《Head First Java》开始，逐步成就了今天的Head First系列。最开始写这本书时，Bert曾与我做过一次长达90分钟的“马拉松”电话交流，帮助我确定这本书的基调，使我得以将思维延伸至极致，让这本书更为出色。如今，这个电话已经过去9个月了，我才刚刚从Bert带给我的震撼和兴奋中恢复过来。

致我的朋友和同事：

再次感谢卡罗理工学院计算与网络系主任Nigel Whyte能够支持我编写另一本书（特别是距上一本书的交付出版还时隔不久）。

我的学生们（游戏开发专业大三学生和软件工程专业大四学生）在过去18个月中曾以不同方式接触过这本书的内容。他们对Python的积极回应以及我在课上采用的方法都对这本书的结构以及最后内容的形成有很大帮助（没错，有些内容正是期末试卷上的题目）。

致我的家人：

我亲爱的家人Deirdre、Joseph、Aaron和Aideen不得不再一次忍受我的牢骚连连、吹胡子瞪眼，还有不时发作的坏脾气（不过，说实在的，与写《Head First Programming》时相比，这一次我发火的次数要少一点了）。完成上一本书之后，我答应“一段时间内”不再变成那个脾气暴躁的形象。可惜这“一段时间”仅仅保持了几个星期而已，我衷心感谢家人们没有因为我的失言集合起来把我扔出门外。如果没有他们的支持，特别是我的妻子Deirdre无尽的爱和支持，这本书绝无可能问世。

一个也不能少：

感谢我的技术审校团队出色的工作，是他们让我没有脱离正轨，保证我所说的准确无误。他们对正确的内容做出确认，对不正确的部分提出质疑，不仅指出哪里有问题，还给出了具体的修改建议。特别是David Griffiths，他是《Head First Programming》的合作者，他做出的技术评论远远超出他的职责范围。虽然这本书封面上没有David的名字，但是书中很多内容都源于他的想法和建议。作为《Head First Python》的技术审校人员，他对这个角色如此热情的投入让我充满敬意，而且永远感激不尽。

目录（概览）

	引子	xxiii
1	初识Python：人人都爱列表	1
2	共享你的代码：函数模块	33
3	文件与异常：处理错误	73
4	持久存储：数据保存到文件	105
5	推导数据：处理数据！	139
6	定制数据对象：打包代码与数据	173
7	Web开发：集成在一起	213
8	移动应用开发：小设备	255
9	管理你的数据：处理输入	293
10	扩展你的Web应用：来真格的	351
11	处理复杂性：数据加工	397
i	其他：（我们没有谈到的）十大问题	435
索引		447

详细目录

引子

你的大脑与Python。你想学些新东西，但是你的大脑总是帮倒忙，它会努力让你记不住所学的东西。你的大脑在想："最好留出空间来记住那些确实重要的事情，比如要避开哪个野生动物，还有裸体滑雪是不是不太好？"那么，如何让你的大脑就范？让它认为如果不知道Python你将无法生存！

谁适合看这本书？	xxiv
我们知道你的大脑在想什么	xxv
元认知	xxvii
由你让你的大脑就范	xxix
重要说明	xxx
技术审校团队	xxxii
致谢	xxxiii

1 初识Python

人人都爱列表

你可能会问："Python有什么与众不同的地方？"如果简短地回答这个问题，那么答案是：很多很多。要想更详细地回答，首先需要指出Python也有很多我们熟悉的东西。类似于所有其他通用编程语言，Python同样有语句、表达式、操作符、函数、模块、方法和类。这些确实都很普通。不过，Python还能提供一样东西，能让程序员（也就是你）的日子更好过一些。下面先从学习列表开始我们的Python之旅。不过，在此之前，还有一个重要的问题需要回答……

Python有什么过人之处？	2
安装Python 3	3
使用IDLE来帮助学习Python	4
有效地使用IDLE	5
处理复杂数据	6
创建简单的Python列表	7
列表就像是数组	9
向列表增加更多数据	11
处理列表数据	15
For循环处理任意大小的列表	16
在列表中存储列表	18
在列表中查找列表	20
复杂数据很难处理	23
处理多层嵌套列表	24
不要重复代码；应当创建一个函数	28
在Python中创建一个函数	29
解决之道：递归！	31
你的Python工具箱	32

The Holy Grail, 1975, Terry Jones & Terry Gilliam, 91 mins

Graham Chapman

Michael Palin, John Cleese, Terry Gilliam, Eric Idle & Terry Jones

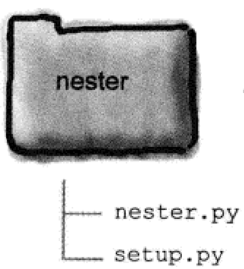
PDG

2 共享你的代码

函数模块

可重用的代码固然不错，不过可共享的模块更棒。通过作为Python模块共享代码，就可以向整个Python社区开放你的代码……共享总是一件好事，不是吗？在这一章中，你将学习如何创建、安装和发布你自己的可共享模块，然后把你的模块加载到Web上的Python软件共享网站，这样所有人都能受益于你的工作。在学习过程中，你还会了解一些与Python函数有关的新技巧。

太好了，所以应该分享	34
函数转换为模块	35
模块无处不在	36
注释代码	37
准备发布	40
构建发布	41
发布速览	42
导入模块并使用	43
Python的模块实现命名空间	45
注册PyPI网站	47
向PyPI上传代码	48
欢迎来到PyPI社区	49
用额外的参数控制行为	52
写新代码之前，先考虑BIF	53
Python会尽力运行你的代码	57
跟踪代码	58
找出哪里出了问题	59
用你的新代码更新PyPI	60
你改变了API	62
使用可选参数	63
模块支持两个API	65
API还是不对	66
模块重获声誉	70
你的Python工具箱	71

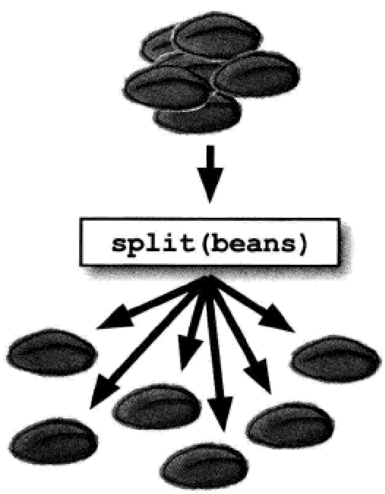


3 文件与异常

处理错误

只是在代码中处理列表数据还不够。你还需要把数据轻松地送入程序。如此说来，不难理解为什么利用Python可以很容易地从文件读取数据。这一点很棒，不过再想想看，与程序之外的数据交互时可能有麻烦……另外还有方方面面的问题在等着给你下绊！出问题时，需要一种策略使你能够从麻烦中脱身，利用Python的异常处理机制来处理异常情况就是这样一种策略，这一章将会介绍这种机制。

程序外部的数据	74
都是文本行	75
进一步查看数据	77
了解你的数据	79
了解你的方法，请求帮助	80
更好地了解你的数据	82
两种截然不同的方法	83
增加额外逻辑	84
处理异常	88
先尝试，然后恢复	89
找出要保护的代码	91
放过错误	93
其他错误呢？	96
增加更多错误检查代码……	97
……或者再增加一层异常处理	98
那么，哪种方法更好呢？	99
大功告成……不过还有一个小问题	101
特定指定异常	102
你的Python工具箱	103

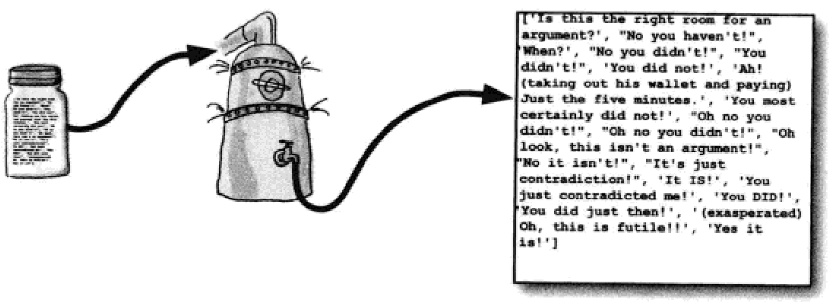


4 持久存储

数据保存到文件

能够处理基于文件的数据确实很棒。但是工作完成时你的数据会怎么样呢？当然，最好把数据保存到一个磁盘文件中，这样就能在以后某个时间再次使用这些数据。将基于内存的数据存储到磁盘上，这正是持久存储的含义。Python支持所有将数据写至文件的常用工具，另外还提供了一些很酷的工具，可以高效地存储Python数据。所以……请翻开下一页，下面开始学习这些内容。

程序生成数据	106
以写模式打开文件	110
发生异常后文件会保持打开！	114
用finally扩展try	115
知道错误类型还不够	117
用with处理文件	120
默认格式对文件并不合适	124
何不修改 print_lol()?	126
“腌制”数据	132
用dump保存，用load恢复	133
使用pickle的通用文件I/O才是上策！	137
你的Python工具箱	138

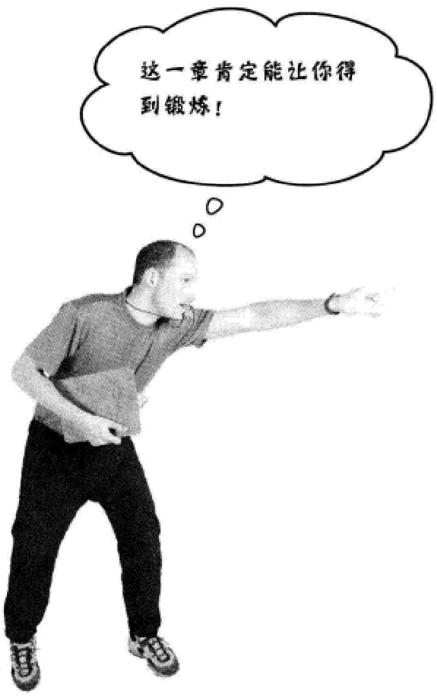


5 推导数据

处理数据！

数据各式各样，有不同的大小、格式和编码。要想有效地处理你的数据，通常需要把它处理并转换为一种常用的格式，以便高效处理、排序和存储。在这一章中，我们会研究Python的一些有助于有效处理数据的工具，让你感受到数据处理的非凡之处。好吧，翻开下一页，别让教练等久了……

Kelly教练需要你的帮助	140
排序有两种方式	144
时间的麻烦	148
推导列表	155
迭代删除重复项	161
用集合删除重复项	166
你的Python工具箱	172

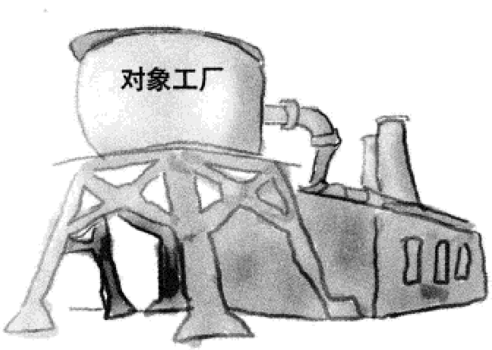


6 定制数据对象

打包代码与数据

你选择的数据结构要与数据匹配，这很重要。而且这个选择将对代码的复杂性带来很大差别。在Python中，尽管列表和集合确实很有用，但这并不是全部。Python还提供了字典，允许你有效地组织数据，可以将数据与名关联而不是与数字关联，从而实现快速查找。当Python的内置数据结构无法胜任时，Python class语句还允许你定义自己的数据结构。这一章就会介绍这些内容。

Kelly又来了（带来一种新的文件格式）	174
使用字典关联数据	178
将代码及其数据打包在类中	189
定义一个类	190
使用class定义类	191
self的重要性	192
每个方法的第一个参数都是self	193
继承Python内置的list	204
Kelly教练相当满意	211
你的Python工具箱	212



7 Web开发

集成在一起

你迟早会希望与很多人分享你的应用。要做到这一点，你有很多选择。可以把你的代码上传到PyPI，发送大量email邮件，把你的代码放在一个CD或USB上，或者只要有人需要，就直接把应用手动安装在他们的计算机上。听上去要做很多工作……更何况这会让人很头疼。另外，如果你开发出代码的下一个最佳版本又会发生什么情况？你将如何管理代码的更新？面对现实吧，要想出一个有创意的借口确实很困难。幸运的是，你根本不用去找借口，只需创建一个Web应用就行了。另外，正如这一章所要展示的，用Python完成Web开发确实非常轻松。

分享是好事 214
可以把你的程序放在Web上 215
Web应用需要做什么？ 218
采用MVC设计Web应用 221
为数据建模 222
查看界面 226
控制你的代码 234
CGI让Web服务器运行程序 235
显示选手列表 236
可怕的404错误！ 242
创建另一个CGI脚本 244
启用CGI跟踪来帮助解决错误 248
一个小改变会让一切大不同 250
你的Web应用妙极了！ 252
你的Python工具箱 253

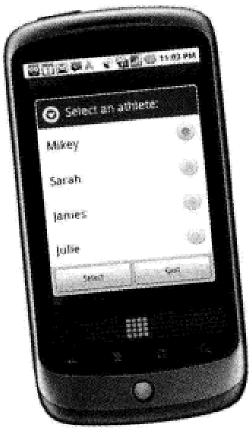


8 移动应用开发

小设备

数据放在Web上就像打开了潘朵拉的盒子。不仅任何人可以从任何地方与你的Web应用交互，而且越来越多的人在通过各种各样的计算设备访问你的Web应用，比如PC、笔记本电脑、平板电脑、掌上电脑，甚至移动电话。现在你不仅需要支持和考虑与Web应用交互的人，还要考虑机器人，也就是能自动完成Web交互的小程序，它们通常只想得到你的数据，并不需要对人类友好的HTML。这一章将在Kelly教练的移动电话上利用Python编写一个应用，来访问Web应用的数据。

世界越来越小	256
Kelly教练在使用Android	257
不用担心Python 2	259
建立开发环境	260
配置SDK和模拟器	261
安装和配置Android脚本环境	262
为SL4A安装增加Python	263
在Android上测试Python	264
定义应用的需求	266
SL4A Android API	274
在Android上选择列表	278
选手数据CGI脚本	281
看起来应该改变数据的类型	284
JSON无法处理你的定制数据类型	285
在真正的手机上运行你的应用	288
配置AndFTP	289
教练对应用大加赞赏	290
你的Python工具箱	291



9 管理你的数据

处理输入

Web和手机并不只是能很好地显示数据。它们也是从用户接收输入的绝妙工具。当然，一旦Web应用接收数据，肯定需要把数据放在某个地方。在决定把数据放在哪里时，你的选择往往会对Web应用带来巨大差别。如果选择得当，Web应用将很易于扩展，否则扩展可能很困难。在这一章中，你将扩展你的Web应用从Web（通过浏览器或从Android手机）接收数据，另外还将了解并改进后台数据管理服务。

你的选手时间应用已经声名远扬	294
使用表单或对话框接收输入	295
创建一个HTML表单模板	296
数据传送到CGI脚本	300
在Android手机上请求输入	304
该更新服务器数据了	308
避免竞态条件	309
需要一个更好的存储机制	310
使用数据库管理系统	312
Python包括SQLite	313
利用Python的数据库API	314
数据库API的相应Python代码	315
小小的数据库设计会带来很大不同	316
定义数据库模式	317
数据是什么样?	318
从pickle向SQLite传输数据	321
为选手指定了什么ID?	322
插入计时数据	323
SQLite数据管理工具	326
SQLite与现有Web应用集成	327
仍然需要名字列表	332
根据ID得到选手的详细信息	333
还需要修改Android应用	342
更新SQLite中的选手数据	348
NUAC非常满意!	349
你的Python工具箱	350

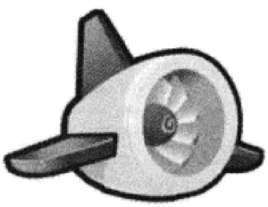


10 扩展你的Web应用

来真格的

你的应用确实很适合放在Web上……除非开始来真格的。总有一天你会走运，你的Web应用可能大获成功。等到那一天，你的Web应用不再每天只有寥寥无几的点击量，可能会达到上千、上万，甚至更多。做好准备了吗？你的Web服务器能处理这么大的负载吗？你又怎么知道它能不能应对？开销有多大？谁来承担费用？你的数据模型能不能扩展到包含数百万的数据项而不会导致应用缓慢如牛？利用Python可以很容易地启动和运行Web应用，而且现在有了Google App Engine，扩展Python Web应用也完全可以轻松实现。

到处都有人看到鲸	352
HFWWG需要自动化	353
用Google App Engine构建Web应用	354
下载和安装App Engine	355
确保App Engine正常工作	356
App Engine使用MVC模式	359
用App Engine对数据建模	360
如果没有视图，模型有什么用？	363
使用App Engine中的模板	364
Django的表单验证框架	368
检查表单	369
控制App Engine Web应用	370
提供选择来限制输入	376
遭遇“死亡白屏”	378
在Web应用中处理POST	379
把数据放在datastore中	380
不要破坏“健壮性原则”	384
接受几乎所有日期和时间	385
看起来你还没有完成	388
有时，最小的改变可能会带来天壤之别……	389
还要捕获用户的Google ID	390
将Web应用部署到Google云	391
HFWWG Web应用已经成功部署！	394
你的Python工具箱	395

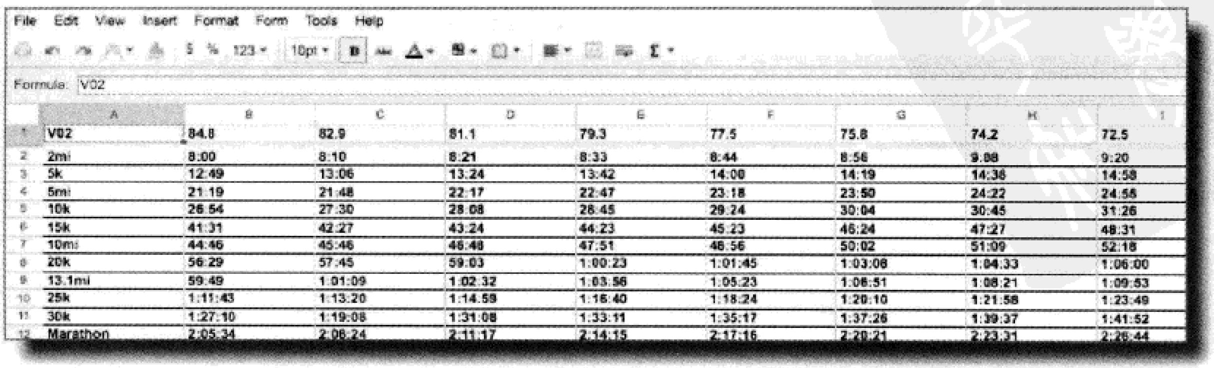


11 处理复杂性

数据加工

能在一个特定领域应用Python确实很棒。不论是Web开发、数据库管理还是移动应用，Python都能帮助你完成任务，顺利地编写解决方案。不过除了这些问题，还存在另外一些问题：它们无法归类或关联到某一领域。这些问题本身很特别，必须用一种有区别的、非常特定的方式来考虑。为这些问题创建预定的软件解决方案正是Python擅长的领域。在最后这一章中，你的Python技能将会更上一层楼，从而能很好地解决这些问题。

下一次跑步有没有合适的目标时间？ 398
那么……有什么问题吗？ 400
从数据开始 401
将各个时间存储为字典 407
预测代码剖析 409
得到用户输入 413
获取输入产生了一个问题…… 414
搜索最接近的匹配 416
时间有问题 418
时间—秒转换模块 419
时间还有问题…… 422
移植到Android 424
你的Android应用就是一堆对话框 425
集成应用…… 429
应用大功告成！ 431
你的Python工具箱 432



	A	B	C	D	E	F	G	H	I
1	V02	84.8	82.9	81.1	79.3	77.5	75.8	74.2	72.5
2	2mi	8:00	8:10	8:21	8:33	8:44	8:56	9:08	9:20
3	5k	12:49	13:06	13:24	13:42	14:00	14:19	14:38	14:58
4	5mi	21:19	21:48	22:17	22:47	23:18	23:50	24:22	24:56
5	10k	26:54	27:30	28:08	28:45	29:24	30:04	30:45	31:26
6	15k	41:31	42:27	43:24	44:23	45:23	46:24	47:27	48:31
7	10mi	44:46	45:46	46:48	47:51	48:56	50:02	51:09	52:18
8	20k	56:29	57:45	59:03	1:00:23	1:01:45	1:03:08	1:04:33	1:06:00
9	13.1mi	59:49	1:01:09	1:02:32	1:03:56	1:05:23	1:06:51	1:08:21	1:09:53
10	25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24	1:20:10	1:21:58	1:23:49
11	30k	1:27:10	1:19:08	1:31:08	1:33:11	1:35:17	1:37:26	1:39:37	1:41:52
12	Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16	2:20:21	2:23:31	2:26:44

i 其他

（我们没有谈到的）十大问题

你已经学到了不少。

不过学习Python是永无止境的。你编写的Python代码越多，就越需要了解新的方法来处理某些事情。你还要掌握新工具和新技术。这本书实在篇幅有限，无法向你全面展示关于Python所需了解的一切。所以，下面给出我们没有谈到的十大问题，你可能希望下一步对这些问题有更多了解。

#1: 使用一个“专业”IDE 436
#2: 处理作用域 437
#3: 测试 438
#4: 高级语言特性 439
#5: 正则表达式 440
#6: 关于Web框架 441
#7: 对象关系映射工具和NoSQL 442
#8: GUI编程 443
#9: 要避免的问题 444
#10: 其他Python书 445



1 初识Python

人人都爱列表



你可能会问："Python有什么与众不同的地方？"

如果简短地回答这个问题，那么答案是：很多很多。要想更详细地回答，首先需要指出Python也有很多我们熟悉的东西。类似于所有其他通用编程语言，Python同样有语句、表达式、操作符、函数、模块、方法和类。这些确实都很普通。不过，Python还能提供一样东西，能让程序员（也就是你）的日子更好过一些。下面先从学习列表开始我们的Python之旅。不过，在此之前，还有一个重要的问题需要回答……

Python有什么过人之处？

很多很多。可以这么说，这本书的目标就是向你展示Python的妙处。



深入研究Python之前，先来完成一些日常工作。

要使用和执行本书中的Python代码，你的计算机上需要有一个Python 3解释器。就像Python的很多其他方面一样，安装这个解释器并不难。当然，这里假设原先尚未安装Python……

安装Python 3

编写和运行Python代码之前，需要确保你的计算机上安装有Python解释器。在本书中，我们将采用Python 3，这是这个语言的新（也是最好）版本。

你的计算机上可能已经安装了某个版本的Python。Mac OS X会预装Python 2，Linux的大多数版本也是如此（也可能预装Python 3）。但Windows有所不同，它未内置任何Python版本。下面来检查你的计算机上是否安装有Python 3。打开一个命令行提示窗口，如果你使用的操作系统是Mac OS X或Linux，可以键入：

```
python3 -V
```

顺便说一句，这是一个大写的"V"。

在Windows上，则要使用下面这个命令：

```
c:\Python31\python.exe -V
```

动手做！

如果你的计算机上没有安装Python 3，可以从*www.python.org*网站针对你喜欢的操作系统下载一个合适的Python版本。

如果使用大写的"V"，会在屏幕上显示Python版本。

如果没有大写的"V"，则会进入Python解释器。

使用quit()命令会退出解释器，返回到操作系统提示符。

```
File  Edit  Window  Help  WhichPython?
$ python3 -V
Python 3.1.2
$
$ python3
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more info.
>>>
>>> quit()
$
```

安装Python 3时，还会得到一个IDLE，这是Python的集成开发环境，尽管简单，但极其有用。IDLE包括一个能够利用颜色突出显示语法的编辑器、一个调试工具、Python Shell，以及一个完整的Python 3在线文档集。

下面先来简单看看IDLE。

使用IDLE来帮助学习Python

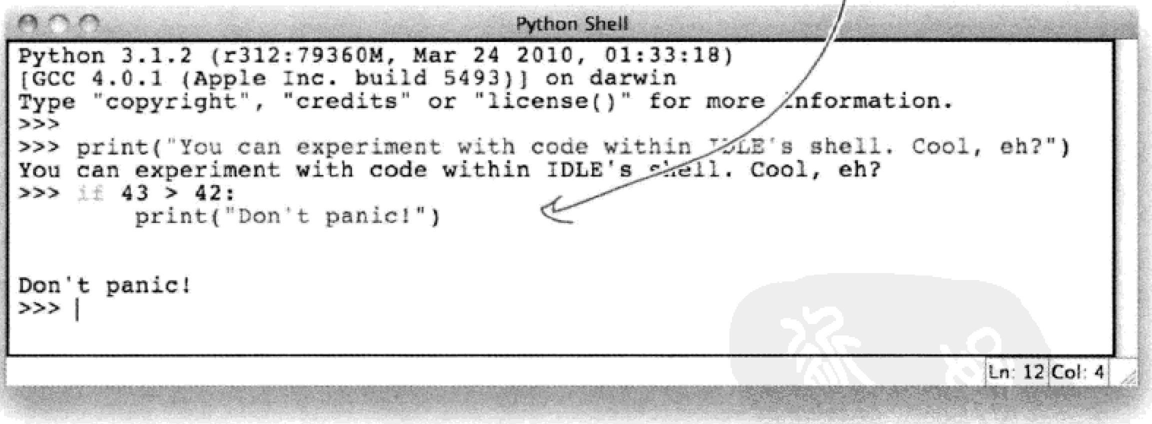
IDLE提供了一个功能完备的代码编辑器，允许你在这个编辑器中编写代码，另外还有一个 Python Shell，可以在其中试验运行代码。本书后面会使用这个代码编辑器，不过要知道，学习Python时，IDLE的shell真的很棒，因为利用它你可以在编写新Python代码的同时尝试运行。

第一次启动IDLE时，会显示“三个大于号”提示符（>>>），可以在这里输入代码。shell得到你的代码语句后会立即执行，并在屏幕上显示生成的结果。

IDLE知道所有Python语法，它会提供“完成提示”（当你使用一个内置函数（如`print()`）时就会弹出这种“完成提示”）。Python程序员通常把内置函数称为BIF（built-in functions）。`print()` BIF的作用是把消息显示到标准输出（通常就是屏幕）。

与其他基于C的语言不同，它们往往使用{和}来界定代码块，Python则利用缩进指示代码块。

在>>>提示符后面输入代码。



可以立即看到结果。

IDLE使用区分颜色的语法来突出显示代码。默认地，内置函数都是紫色，字符串是绿色，Python语言的关键字（如`if`）为橙色。生成的所有结果显示为蓝色。如果你不喜欢这些颜色选择，也不用担心。只需调整IDLE的首选项就可以很容易地改变颜色选择。

另外，IDLE也很清楚Python的缩进语法（Python要求代码块缩进）。刚开始使用Python时，你可能很难适应这一点，不过IDLE可以减轻你的负担，它会根据需要自动缩进。

IDLE了解Python的语法，并会帮助你遵循Python的缩进规则。

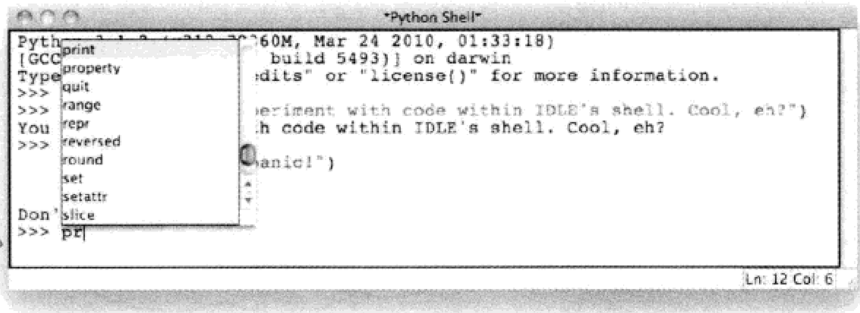
有效地使用IDLE

IDLE提供了大量特性，不过只需了解其中一小部分就能很好地使用IDLE。

TAB完成

先键入一些代码，然后按下TAB键。IDLE会提供一些建议，帮助你完成这个语句。

这是我的机器上的IDLE界面。看起来可能会与你的IDLE界面稍有不同，不过差别不会太大（没错，确实这么丑）。



在>>>提示符后面键入"pr"，然后按下TAB，可以看到IDLE给出的一个列表，其中包括完成这个命令的一组建议。

回退代码语句

按下Alt-P，可以回退到IDLE中之前输入的代码语句，或者按下Alt-N可以移至下一个代码语句（如果有的话）。可以利用这两个按键组合在IDLE中已输入的所有代码之间快速转换，根据需要重新执行其中的任何代码语句。

Alt-P表示"前一个"（Previous）

Alt-N表示"下一个"（Next）

除非你在使用Mac，这种情况下则要使用Ctrl-P和Ctrl-N。

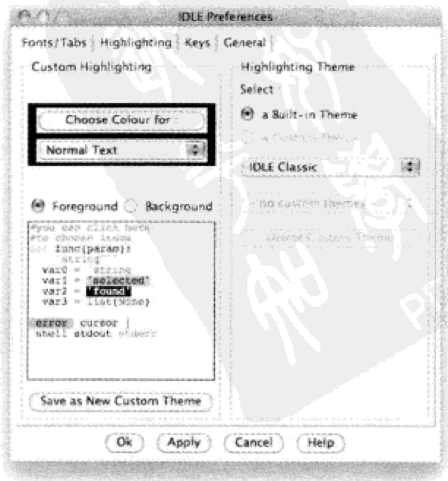
编辑回退的代码

一旦回退代码语句，还可以进行编辑，并使用箭头键切换语句。可以编辑之前输入的任何语句，甚至是跨多行的代码语句。

调整IDLE的首选项

IDLE的首选项对话框允许你根据个人口味调整IDLE的默认行为。有4个设置标签页可以调整。可以控制字体和tab行为，改变突出显示语法所用的颜色，调整某些按键组合的行为，还可以改变IDLE的启动设置。所以，如果你确实喜欢鲜艳的粉红色字符串，IDLE会赋予你这个能力，可以改变代码在屏幕上如何显示。

按你心中所想来调整IDLE。

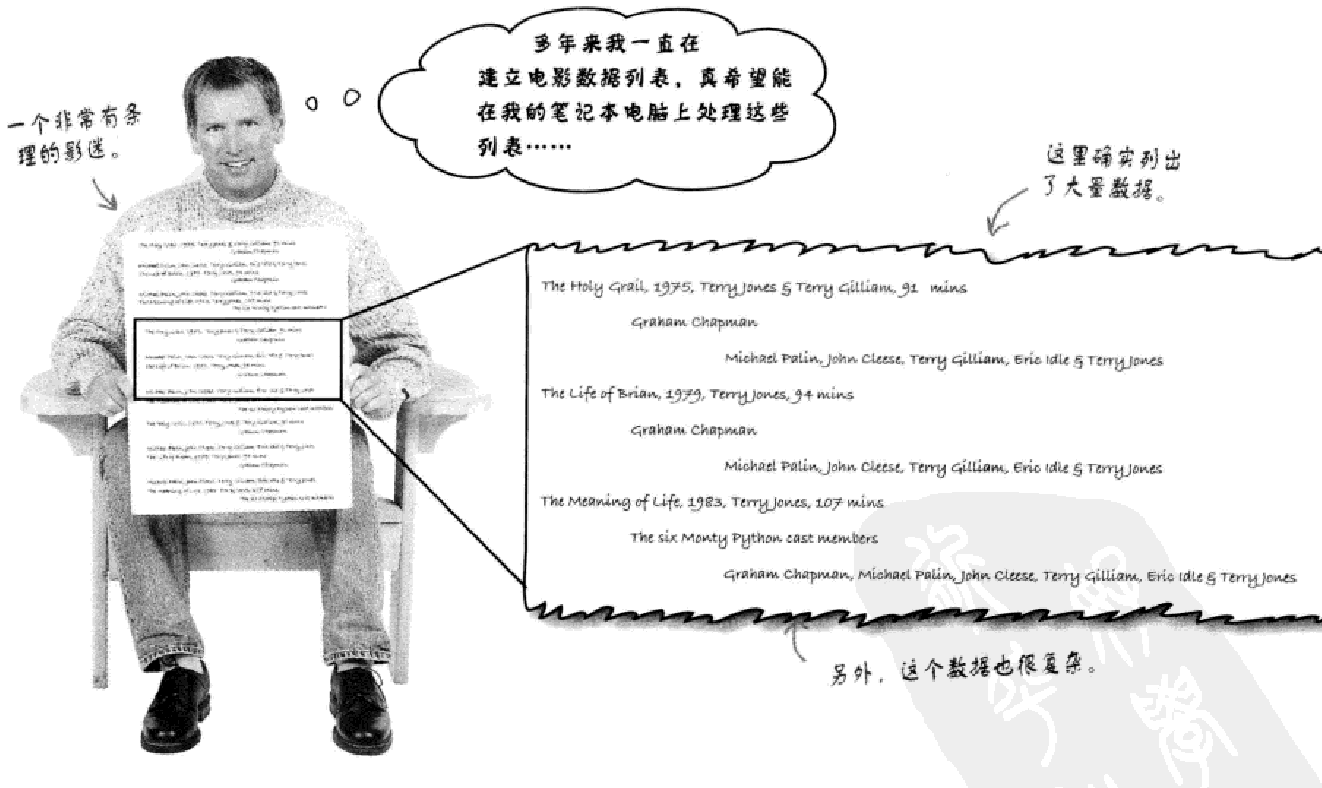


处理复杂数据

任何值得着手创建的程序都必然会处理数据。有些时候，数据很简单也很直接——很容易处理。但还有些情况下，所要处理的数据不论从结构还是含义上讲都很复杂，这就要求你花些工夫把它完全搞清楚，另外编写代码来处理这些数据更是需要费些心思。

为了降低复杂性，通常可以把数据组织为列表：可以有顾客列表、朋友列表、购物清单和待办事项清单等。由于用列表组织数据如此常见，Python中可以很容易地利用代码创建和处理列表。

在学习如何用Python创建和处理列表数据之前，先来看一些复杂的数据。



乍一看，这组数据确实相当复杂。不过，这组数据似乎符合某种结构：最前面一行给出一组电影基本信息，接下来一行是主要演员，再后面第三行列出了这部电影的配角。

看起来这个结构是可以处理的……

创建简单的Python列表

先从下面这个简单的电影片名列表入手，在此基础上逐步深入：

The Holy Grail

The Life of Brian

The Meaning of Life

一个简短的Monty Python电影列表。

下面是同一个列表，不过这一次用Python能理解的方式来写：

```
movies = ["The Holy Grail",
          "The Life of Brian",
          "The Meaning of Life"]
```

为了把人可读的列表转换为Python可读的列表，需要遵循以下4个步骤：

1. 在数据两边加引号，将各个电影名转换为字符串。
2. 用逗号将列表项与下一项分隔开。
3. 在列表的两边加上开始和结束中括号。
4. 使用赋值操作符（=）将这个列表赋至一个标识符（以上代码中的movies）。

完全可以把创建列表的代码都放在一行上（当然，前提是假设你有足够的空间，全部代码都能在一行上放下）：

```
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
```

这样也是可以的。



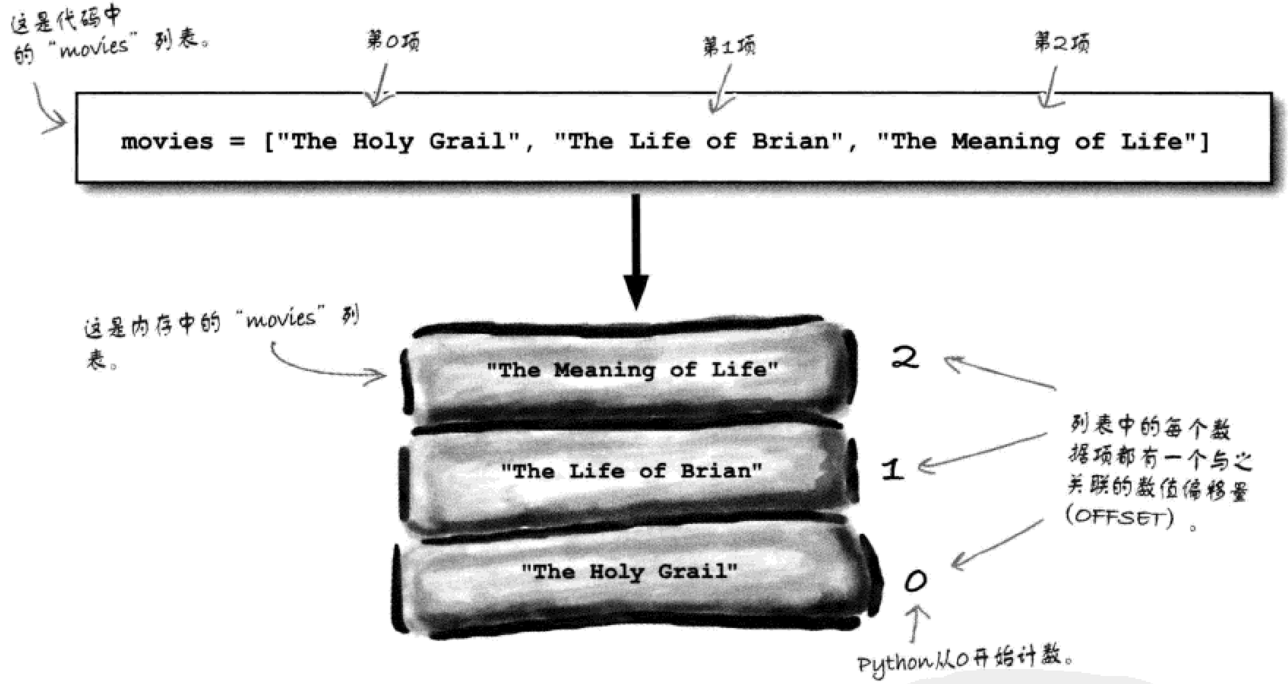
大可不必，因为Python的变量标识符没有类型。

很多其他编程语言坚持要求代码中使用的每一个标识符都必须声明有类型信息。但是这对Python并不适用：标识符只是名字，可以指示某个类型的数据对象。

可以把Python的列表想成是一个高层集合。对于列表来说，数据项的类型并不重要。当然可以说你的电影列表是一个“字符串集合”，不过Python并不需要知道这一点。Python所要知道的只是你需要一个列表，而且已经为它指定了一个名字，另外这个列表中包含有一些数据项。

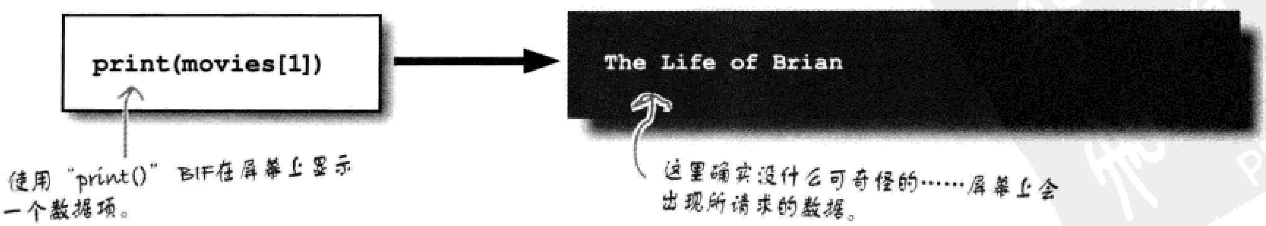
列表就像是数组

在Python中创建一个列表时，解释器会在内存中创建一个类似数组的数据结构来存储数据，数据项自下而上堆放（形成一个堆栈）。类似于其他编程语言中的数组技术，堆栈中的第一个槽编号为0，第二个槽编号为1，第3个编号为2，依此类推：



使用中括号记法访问列表数据

像数组一样，可以使用标准的中括号偏移量记法来访问一个列表槽中的数据项：



下面利用IDLE进一步学习列表是如何工作的。

An IDLE Session

Python中的列表看起来可能很像数组，不过还不只如此：列表是完备的Python集合对象。也就是说，列表通过列表方法的形式提供了一些现成的功能。

下面来了解Python列表的一些方法。打开IDLE，在>>>提示符后面输入如下代码。应该能看到与这里相同的输出。

首先定义一个名字列表，然后使用`print()` BIF在屏幕上显示这个列表。接下来，使用`len()` BIF得出列表中有多少个数据项，然后再访问并显示第2个数据项的值：

```
>>> cast = ["Cleese", 'Palin', 'Jones', "Idle"]
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle']
>>> print(len(cast))
4
>>> print(cast[1])
Palin
```

对一个BIF的结果再调用另一个BIF是完全可以的。

创建了列表之后，可以使用列表方法在列表末尾增加一个数据项（使用`append()`方法），或者从列表末尾删除数据（使用`pop()`方法），还可以在列表末尾增加一个数据项集合（利用`extend()`方法）：

```
>>> cast.append("Gilliam")
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
>>> cast.pop()
'Gilliam'
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle']
>>> cast.extend(["Gilliam", "Chapman"])
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam', 'Chapman']
```

使用常用的“.”（点记法）来调用方法。

这是另一个列表：各数据项之间用逗号隔开，整个列表用中括号括起。

最后，在列表中找到并删除一个特定的数据项（使用`remove()`方法），然后在某个特定的位置前面增加一个数据项（使用`insert()`方法）：

```
>>> cast.remove("Chapman")
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
>>> cast.insert(0, "Chapman")
>>> print(cast)
['Chapman', 'Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
```

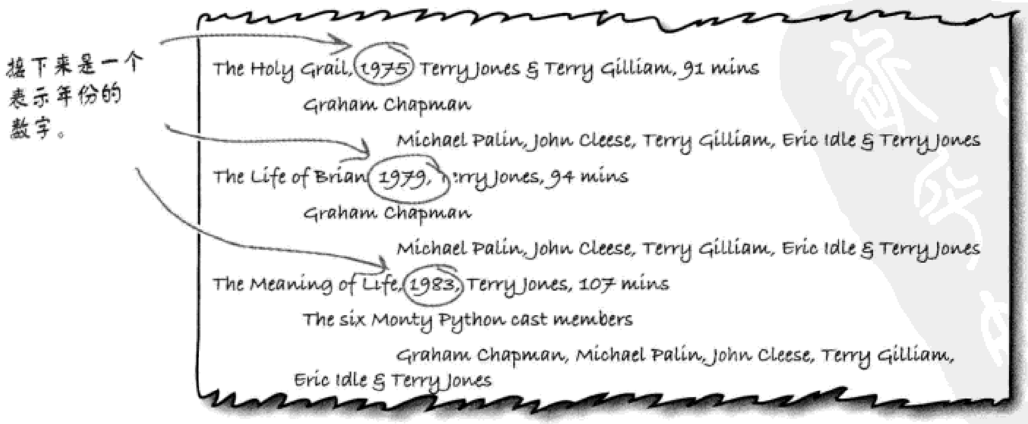
完成所有这些调用之后，最终会得到Monty Python影片Flying Circus的演员表！

向列表增加更多数据

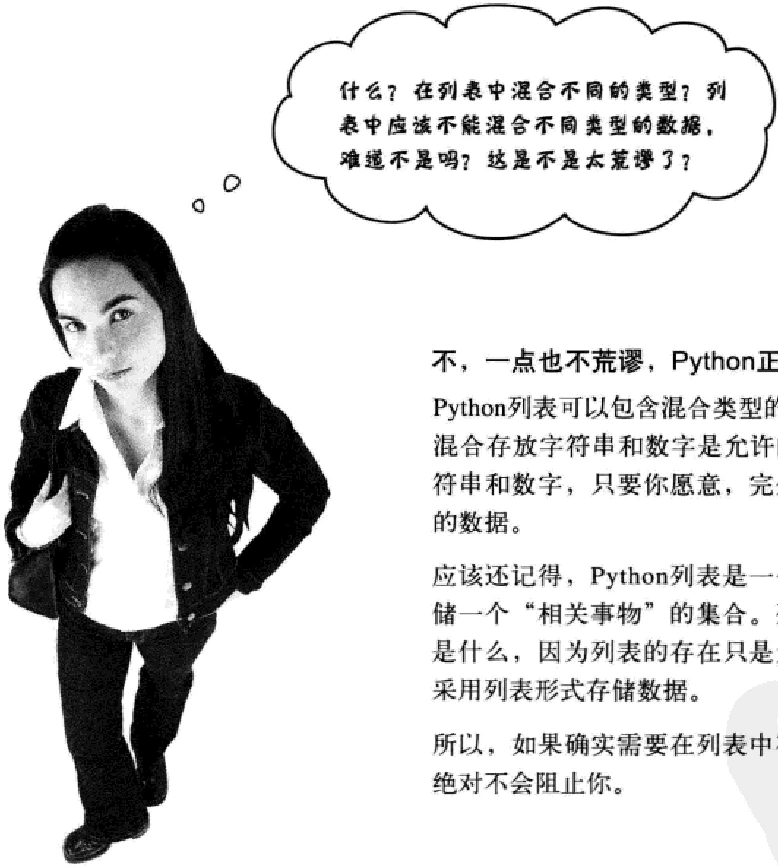
创建了电影片名列表之后，现在需要为它增加这个影迷的更多复杂数据。这里可以做一个选择：



这两种策略都是可行的。至于哪一种方法更合适，这取决于你打算做什么。下面回忆一下这个影迷的数据是怎样的：



向列表增加的下一个数据是一个数字（表示影片发行的年份），它必须插入到各个影片片名后面。下面就来做这个工作，看看会发生什么。



不，一点也不荒谬，Python正是这样做的。

Python列表可以包含混合类型的数据。在同一个Python列表中混合存放字符串和数字是允许的。实际上，不光可以混合字符串和数字，只要你愿意，完全可以在列表中存储任意类型的数据。

应该还记得，Python列表是一个高层集合，原本设计为要存储一个“相关事物”的集合。列表并不关心这些事物的类型是什么，因为列表的存在只是为了提供一种机制，从而可以采用列表形式存储数据。

所以，如果确实需要在列表中存储混合类型的数据，Python绝对不会阻止你。



下面花点时间试试看，在这种情况下向列表增加数据应当采用哪种策略。
给定以下创建列表的代码：

```
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
```

1 得出向以上列表插入年份数值数据所需的Python代码，改变列表，使之最终为以下形式：

```
["The Holy Grail", 1975, "The Life of Brian", 1979, "The Meaning of Life", 1983]
```

在这里写出你的插入代码。→

2 现在用所需的数据从头重新创建列表，请写出重新创建列表所需的代码：

在这里写出重新创建列表的代码。→

对于这种情况，你认为以上两种方法中哪一个更好（请把你的答案圈出来）？

1 或 2

Exercise Solution

下面花点时间试试看，在这种情况下向列表增加数据应当采用哪种策略。
给定以下创建列表的代码：

```
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
```

❶ 得出向以上列表插入年份数值数据所需的Python代码：

```
movies.insert(1, 1975)
movies.insert(3, 1979)
movies.append(1983)
```

将第1个年份插入到第2个列表项前面。

将第2个年份插入到第4个列表项前面。

你算对了吗？第一次插入之后，列表扩大了，所以在计算第二次插入的位置时需要考虑到这一点。

然后把最后一个年份追加到列表末尾。

❷ 还要写出用所需数据从头重新创建列表的相应Python代码：

```
movies = ["The Holy Grail", 1975,
          "The Life of Brian", 1979,
          "The Meaning of Life", 1983]
```

将所有数据赋至"movies"标识符。这会取代原来存储的数据。

对于这种情况，你认为以上两种方法中哪一个更好？（圈出你的答案）。

❶ 或 **(❷)**

没错，在这里第2种方法看起来更好一些……我的意思是，对于这样一个小列表来说，第2种方法更可取。而且，第2种方法不需要做麻烦的计算。

处理列表数据

通常需要迭代处理列表，在这个过程中对每一个数据项完成某个动作。当然，可能经常这样做（这种做法可行，但是不具有伸缩性）：

定义一个列表，并用两个电影片名填充列表项。

```
fav_movies = ["The Holy Grail", "The Life of Brian"]

print(fav_movies[0])
print(fav_movies[1])
```

在屏幕上显示各个列表项的值。

这是处理列表的代码。

这个代码会得到我们预期的结果，使列表的数据都出现在屏幕上。不过，如果以后修改代码，在列表中再增加一个喜欢的电影，以上处理列表的代码就无法达到预期了，因为这个代码没有提到第3个数据项。

这有何难，只需要再增加一个`print()`语句，对不对？

没错，对于一个新增的电影，再加一个`print()`语句是可以的，但是，如果需要再增加上百部喜欢的电影呢？这个问题的规模肯定会让你心生畏惧，因为另外增加所有这些`print()`语句实在太烦琐，你不会愿意这样做，肯定会想方设法逃避。

该迭代了

处理每一个列表项是一个相当常见的需求，所以Python通过提供内置的`for`循环可以非常方便地做到这一点。考虑以下代码，这里重写了前面的代码来使用一个`for`循环：

定义一个列表，并像前面一样填充这个列表。

```
fav_movies = ["The Holy Grail", "The Life of Brian"]

for each_flick in fav_movies:
    print(each_flick)
```

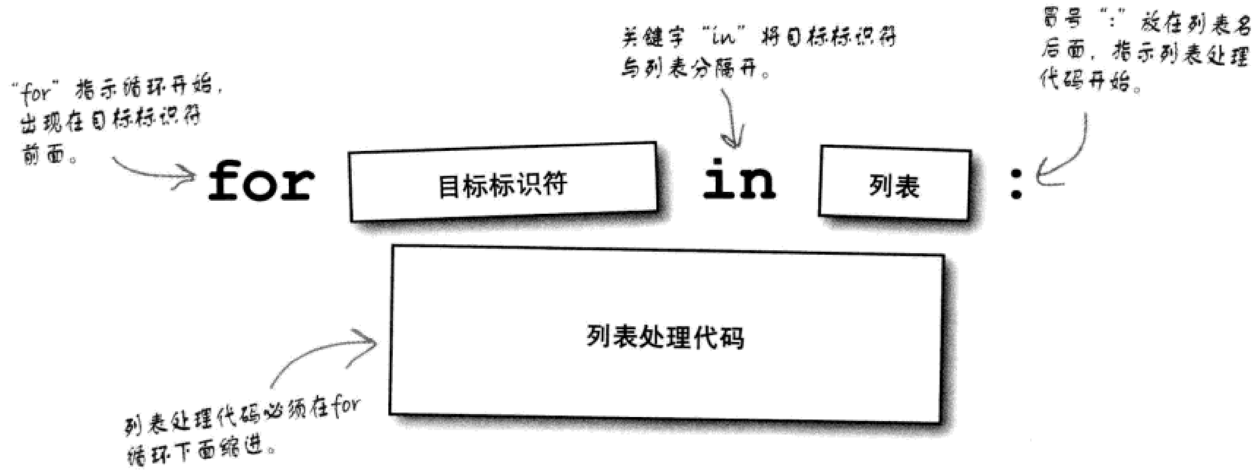
使用"for"迭代处理列表，在这个过程中将各个列表项的值显示在屏幕上。

这是处理列表的代码，这里使用了一个for循环。

使用for循环是可伸缩的，适用于任意大小的列表。

For循环处理任意大小的列表

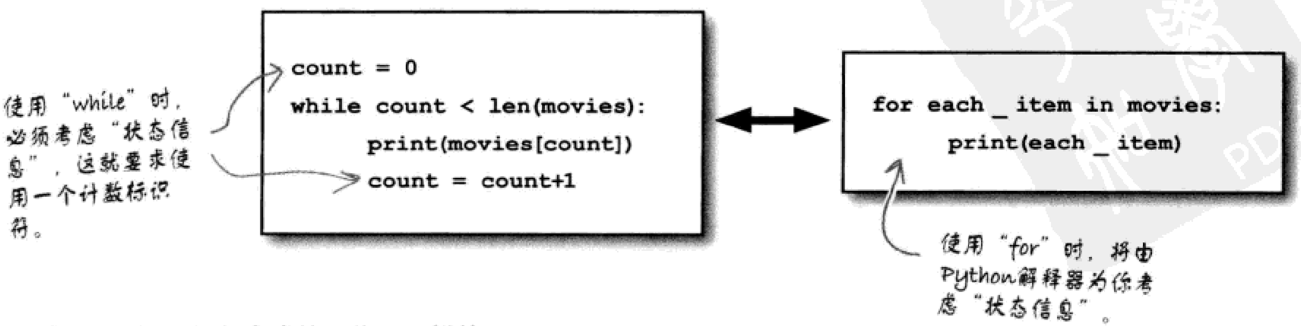
Python的for循环就是为了处理列表和Python中的其他迭代结构。列表是Python中最常用的迭代数据结构，需要迭代处理一个列表时，最好使用for循环：



列表处理代码被Python程序员称为"组"（suite）。

目标标识符（target identifier）类似于代码中的任何其他名。迭代处理列表时，相应的会把列表中的各个数据值分别赋至目标标识符。这说明，每次执行循环代码时，目标标识符都会指示一个不同的数据值。循环会一直迭代，直到处理完列表的所有数据（不论列表有多大或者多小）。

除了使用for，还有一种候选方法，可以使用while循环编写迭代代码。考虑以下两个Python代码段，它们都完成相同的动作：



以上while和for语句完成的工作是一样的。

there are no Dumb Questions

问：这么说……迭代处理一个列表时，是不是总要用for而不是while？

答：对，除非你有非常充分的理由使用while循环（或者需要while循环提供的额外控制）。for循环会负责从列表起始位置开始，一直处理到列表末尾。如果使用for，几乎不太可能遭遇“大小差1”错误。但使用while循环时可就不会这么走运了。

问：那么，列表并不真的像数组，因为列表能做的事情要多得多，是这样吗？

答：没错……你可以用标准的中括号记法访问列表中的单个数据项，从这一点来讲，列表和数组确实很像，不过你已经看到，Python的列表可以做更多的事情。在Head First Labs，我们喜欢把列表认为是“打了激素的数组”。

问：只有Python 3中有列表，是吗？

答：不对。尽管Python 3中确实对列表增加了一些改进，不过Python 2也有列表。到目前为止，你所学到的关于列表的内容对Python 3和Python 2中的列表都适用。

问：为什么我们要使用Python 3？Python 2到底有什么问题？看起来很多程序员都在使用Python 2。

答：确实有相当多的程序员在使用Python 2，不过Python 3才是Python发展的未来。当然，要让整个Python社区完全转向Python 3并不是一蹴而就的事情，所以在可预知的将来，仍会有大量项目继续使用Python 2。尽管Python 2目前占据主导，但在Head First Labs，我们认为Python 3中的新特性确实很妙，很值得“追加投入”来深入学习。不用担心，如果你了解Python 2，Python 3也很容易掌握。

问：我看到Python的列表可以随需要伸缩，它们肯定不支持越界检查，对不对？

答：是这样的，列表可以伸缩，从这一点来讲，列表是动态的，不过它们并没有魔力，因为不能访问一个根本不存在的数据项。如果你试图访问一个不存在的数据项，Python会给出一个`IndexError`作为响应，这就表示“越界”。

问：那些奇怪的Monty Python引用到底是什么？

答：哈，你发现了？看起来Python创始人Guido van Rossum设计这个新的编程语言时肯定在看Monty Python电视剧的剧本。Guido需要为这个新语言找一个名字，他有点开玩笑地选择了“Python”（这也可能只是一个“谣传”）。

问：我是不是需要了解Monty Python才能理解这些例子？

答：大可不必，不过在官方的Python文档中指出：“如果你了解Monty Python会有帮助”。但不用担心，即使你从来没有听说过Monty Python，也一样能过得很好。

问：我注意到，有些字符串用双引号引起来，而另外一些却用单引号引起来。这有什么区别吗？

答：没有任何区别。Python中，单引号和双引号都可以用来创建字符串。对此只有一个规则，这就是如果字符串前面使用了某个引号（单引号或双引号），那么字符串后面也要用同样的引号；不能在字符串前后混合使用不同的引号。你可能已经看到，IDLE在shell中显示字符串时使用了单引号。

问：如果我需要在一个字符串中嵌入一个双引号该怎么做呢？

答：你有两个选择：可以像这样对双引号转义：`\"`，或者用单引号引起这个字符串。

问：可以用任意字符来命名标识符吗？

答：不行。与大多数其他编程语言一样，创建名字时，Python也有一些必须遵循的规则。名字可以以一个字母字符或一个下划线开头，接下来可以包括任意个字母字符、数字和/或下划线。不允许有奇怪的字符（如`%$£`），你肯定希望使用在代码上下文中有意义的名字。类似`members`、`the_time`和`people`等名字就比`m`、`t`和`p`好得多，不是吗？

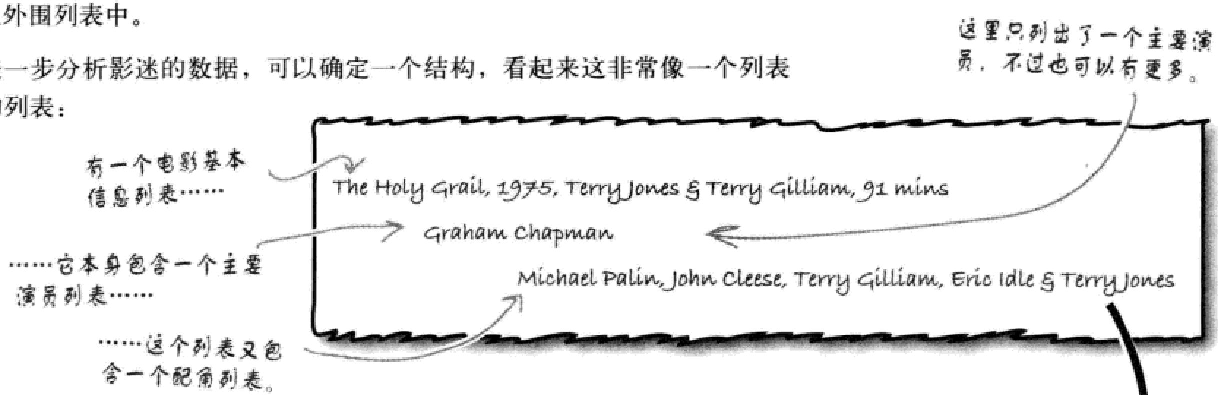
问：确实，好的命名实践通常很重要。那么大小写敏感性呢？

答：嗯，Python属于“敏感型”，因为Python代码区分大小写。这说明，`msg`和`MSG`是两个不同的名字，所以一定要当心。Python（和IDLE）会帮助解决可能因此出现的问题。例如，只有当标识符已经赋值后才能在代码中使用；未赋值的标识符会导致运行时错误。这说明，如果本来想键入`msg`，但实际上误写作`mgs`，你会很快发现问题，因为Python会指出代码存在一个*NameError*错误。

在列表中存储列表

我们已经看到，列表可以存放混合类型的数据。不过还有更棒的：列表能存放任何东西的集合，这也包括其他列表。只需根据需要把内列表嵌在外围列表中。

进一步分析影迷的数据，可以确定一个结构，看起来这非常像一个列表的列表：



在Python中，稍做努力甚至毫不费力就可以把这个真实的数据列表转换为代码。只需记住，每个列表都是一个列表项集合，各列表项相互之间用逗号隔开，另外列表要用中括号括起。当然，任何列表项本身也可以是另一个列表：

第一个外列表的起始位置。

```
movies = [
    "The Holy Grail", 1975, "Terry Jones & Terry Gilliam", 91,
        ["Graham Chapman",
            ["Michael Palin", "John Cleese", "Terry Gilliam", "Eric Idle", "Terry Jones"]]]
```

所有列表都在这里结束。

第二个内列表"movies[4]"的起始位置。

第3个下一级内列表"movies[4][1]"的起始位置。

看起来有点怪异……不过只要记住前面有3个开始中括号，所以肯定还有3个结束中括号，这样就不奇怪了。

所以，列表中嵌套列表是可以的，另外列表中嵌套的列表还可以嵌套下一级列表（前面的代码示例就是如此）。实际上，利用Python，完全可以在列表中嵌套任意多层的列表。可以使用每个列表自己的列表方法来管理，并使用中括号记法来访问：

```
print(movies[4][1][3])
```

```
Eric Idle
```

嵌套在某个列表中的列表，而该列表本身又嵌套在另一个列表中。

Eric嵌套得这么深，所以虽然名字里有Idle（空闲），但他肯定不会很闲☺。

An IDLE Session

创建一个包含另一个列表的列表，这个工作很简单。不过，如果你想使用本章前面的for循环来处理一个包含另一个列表（或多个列表）的列表，会怎么样呢？

下面使用IDLE来看会发生什么。首先在内存中为“The Holy Grail”创建电影数据列表，并在屏幕上显示，然后用for循环处理这个列表：

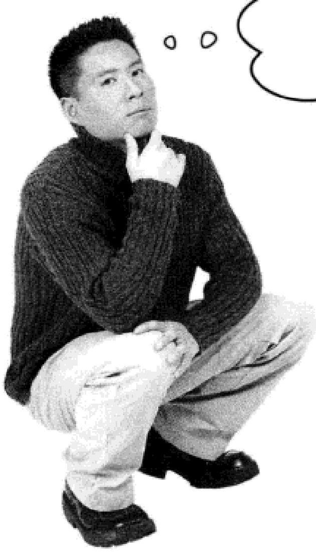
```
>>> movies = ["The Holy Grail", 1975, "Terry Jones & Terry Gilliam", 91,
                ["Graham Chapman", ["Michael Palin", "John Cleese",
                    "Terry Gilliam", "Eric Idle", "Terry Jones"]]]
>>> print(movies)
['The Holy Grail', 1975, 'Terry Jones & Terry Gilliam', 91, ['Graham Chapman', ['Michael Palin',
'John Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry Jones']]]
>>> for each_item in movies:
        print(each_item)


The Holy Grail
1975
Terry Jones & Terry Gilliam
91
['Graham Chapman', ['Michael Palin', 'John Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry Jones']]
```

已经在内存中创建这个三重嵌套的列表。

“for”循环只打印外列表的各个数据项。

嵌套在内列表中的下一层内列表会原样打印。



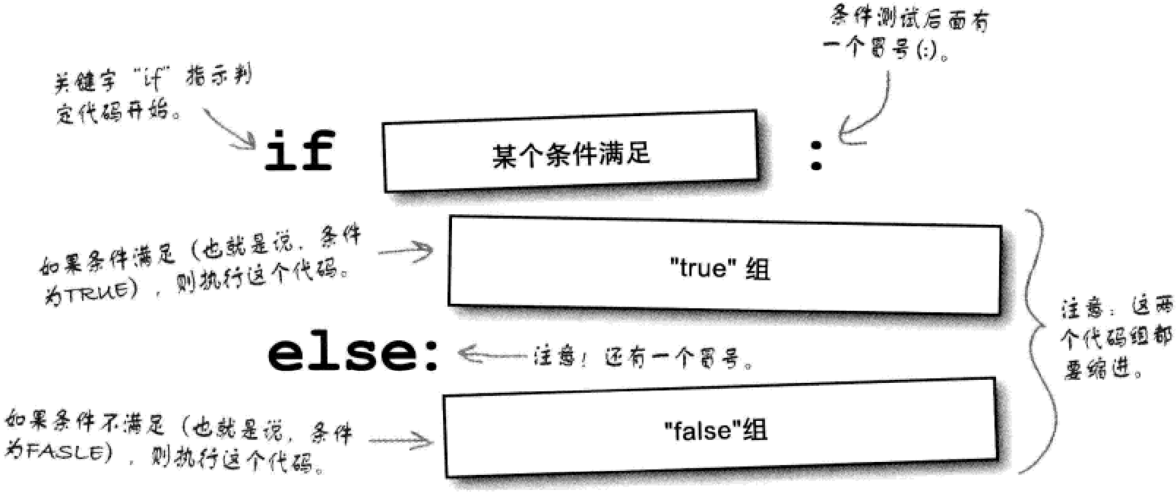
没错，你说对了：这个循环代码并不完备。

目前，循环中的代码只是打印各个列表项，它在某个槽中找到一个列表时，只会在屏幕上把这个列表整个显示出来。毕竟，对于外围列表来说，内列表只是外列表中的一个列表项。这里我们需要一种机制来发现列表中的某一项实际上是另一个列表，并采取适当的行动。

听起来有点困难。不过Python能帮忙吗？

在列表中查找列表

每次处理列表中的一项时，都要查看这一项本身是不是另一个列表。如果这一项确实是一个列表，那么在处理外列表中的下一项之前，先要处理这个列表。Python中确定何时做什么可以采用我们熟悉的**if… else…** 模式：



毫不奇怪，Python中的`if`会按我们预期的那样工作。不过，要检查什么条件呢？你需要一种方法来确定当前处理的列表项本身是不是一个列表。幸运的是，Python内置有一个BIF可以在这方面提供帮助：这个BIF就是`isinstance()`。

`isinstance()` BIF确实很酷，它允许检查某个特定标识符是否包含某个特定类型的数据：

An IDLE Session

下面使用IDLE shell简单了解`isinstance()`如何工作：

```
>>> names = ['Michael', 'Terry']
>>> isinstance(names, list)
True
>>> num_names = len(names)
>>> isinstance(num_names, list)
False
```

创建一个简短的列表，并把它赋至一个标识符。

询问"names"是否是一个列表（确实是）。

将一个数赋至一个标识符。

询问"num_names"是否是一个列表（这不是列表）。

这里指示一个Python类型。在这里，类型为"list"。

Exercise

下面是当前的列表处理代码。你的任务是使用一个if语句和`isinstance()` BIF重写这个代码，适当处理列表从而能够显示嵌套列表。

```
for each_item in movies:
    print(each_item)
```

在这里编写你的新代码。 →

....

....

....

....

....

....

....

....

....

....

there are no Dumb Questions

问： Python中有很多这样的BIF吗？

答： 对，根据最新统计结果，Python 3中有70多个BIF。

问： 70多个！我怎么才能记住那么多BIF？更何况还要知道它们分别做什么！

答： 不用操心记住所有这些BIF。可以让Python帮你。

问： 怎么帮呢？

答： 在Python或IDLE shell中，键入`dir(__builtins__)`可以看到Python提供的内置方法列表（顺便说一句，"builtins"前面和后面分别有两个下划线字符）。shell会给出一个庞大的列表。试试看。所有那些小写单词都是BIF。要查看某个BIF做什么，比如说`input()`，可以在shell中键入`help(input)`，就会得到这个BIF的功能描述。

问： 为什么有这么多BIF？

答： 为什么不呢？由于Python提供了大量内置功能，这就意味着你可以少写代码。这就是Python的"功能齐全"观点：Python已经包含足够多的内置功能，使你能完成大多数工作，而不必依赖第三方的代码。除了大量BIF，你还会发现Python的标准库很丰富，包含大量特性等着你发掘利用。

Exercise Solution

下面是当前的列表处理代码。你的任务是使用一个if语句和isinstance() BIF重写这个代码，适当处理列表从而能够显示嵌套列表。

```
for each_item in movies:
    print(each_item)
```

```
for each_item in movies:
    if isinstance(each_item, list):
        for nested_item in each_item:
            print(nested_item)
    else:
        print(each_item)
```

像从前一样处理"movies"列表。

需要检查当前列表项是不是一个列表。

内循环需要一个新的目标标识符。

如果这是一个列表，使用另一个"for"循环处理这个嵌套列表。

如果外围列表的当前项不是一个列表，则在屏幕上显示这一项。

有没有正确地缩进代码？

An IDLE Session

下面使用IDLE查看这个代码会不会让屏幕上显示的输出有所改变：

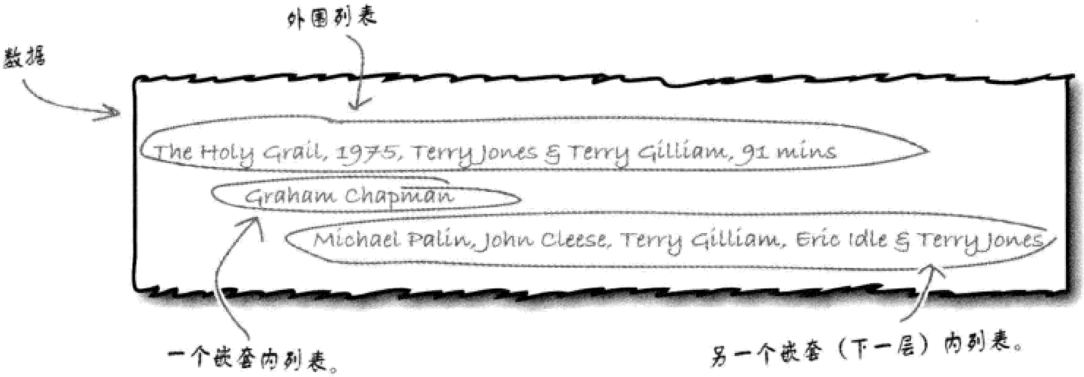
```
>>> for each_item in movies:
        if isinstance(each_item, list):
            for nested_item in each_item:
                print(nested_item)
        else:
            print(each_item)

The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
['Michael Palin', 'John Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry Jones']
```

这样好一点了，不过改进不大……这里还有一个嵌套列表没有得到正确的处理。

复杂数据很难处理

影迷的数据很复杂。下面再来看这个数据的一个子集，并查看处理这个数据的Python代码。



你的代码。

处理外围列表。

处理嵌套内列表。

```
for each_item in movies:
    if isinstance(each_item, list):
        for nested_item in each_item:
            print(nested_item)
    else:
        print(each_item)
```



BRAIN POWER

你能找出现在的Python代码有什么问题吗？你认为需要如何修改代码，才能正确地处理这个影迷的数据？

处理多层嵌套列表

数据和你的代码并不一致。

影迷的数据是一个列表，其中包含一个嵌套列表，这个嵌套列表本身又包含一个嵌套列表。问题在于，你的代码只知道如何处理嵌套在一个外围列表中的列表。

当然，这个问题可以这样来解决：再增加一些代码来处理另外嵌套的列表。通过查看现在的代码，很容易看出需要重复的代码：

```
for each_item in movies:
    if isinstance(each_item, list):
        for nested_item in each_item:
            print(nested_item)
    else:
        print(each_item)
```

这段代码处理一个嵌套列表。

需要把重复的代码放在这里。

代码的下一个迭代是这样的。

```
for each_item in movies:
    if isinstance(each_item, list):
        for nested_item in each_item:
            if isinstance(nested_item, list):
                for deeper_item in nested_item:
                    print(deeper_item)
            else:
                print(nested_item)
    else:
        print(each_item)
```

重复的代码取代"print()"语句，又引入另一个目标标识符，名为"deeper_item"。

注意：在代码中，每个"if"都需要有一个与之关联的"else"。

PDG

An IDLE Session

下面再次使用IDLE测试代码的最后一个迭代：

```
>>> for each_item in movies:
        if isinstance(each_item, list):
            for nested_item in each_item:
                if isinstance(nested_item, list):
                    for deeper_item in nested_item:
                        print(deeper_item)
               else:
                   print(nested_item)
        else:
            print(each_item)


The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
Michael Palin
John Cleese
Terry Gilliam
Eric Idle
Terry Jones
```

处理一个嵌套很深的列表，它位于另一个嵌套列表中，后者本身嵌套在外围列表中。

成功了！这一次，可以在屏幕上看到所有列表数据。



会有更多列表数据和更多Python代码。

这个数据必须作为另一个嵌套列表，嵌在原本层次已经很深的配角嵌套列表中。这是可以的，尽管一想到这是一个“列表中的列表中的列表中的列表”确实让人很头疼！修改这个代码很简单，只需要再增加一个for循环和一个if语句。

听起来还不算太麻烦，是不是？



再增加另一个嵌套循环确实让人很痛苦。

数据越来越复杂（这些“列表中的列表中的列表中的列表”简直能让人崩溃），相应地，你的代码也变得过于复杂（for循环中的for循环中还嵌套有一个for循环，真是让人头都炸了）。一般来讲，过于复杂的代码几乎都不是好东西……

如果能有一个高效的方法处理列表，最好使用一种能少写代码的技术（而不是让代码更多），那该多好，这是不是做梦？不过我知道这只是异想天开吧……



PDG

不要重复代码；应当创建一个函数

再来看我们目前创建的代码（为了不至于让你崩溃），我们已经为你修改了这段代码，它还会处理下一个更深层次的嵌套列表。注意到什么没有？

这段代码与下面这段代码基本上是一样的……

```
for each_item in movies:
    if isinstance(each_item, list):
        for nested_item in each_item:
            if isinstance(nested_item, list):
                for deeper_item in nested_item:
                    if isinstance(deeper_item, list):
                        for deepest_item in deeper_item:
                            print(deepest_item)
                    else:
                        print(deeper_item)
            else:
                print(nested_item)
    else:
        print(each_item)
```

……这段代码实际上与下面这段代码也是一样的……

……而它与这段代码也相差无几。

这4个语句之间都没有太大差别！

这段代码开始有点恐怖了……

现在你的代码包含了大量重复代码。看起来也很乱（尽管它对应着影迷修改后的数据）。所有这些嵌套的**for**循环很难阅读，要确保**else**代码组与相应的**if**语句关联甚至更困难。

肯定还有更好的办法……但是究竟该怎么做呢？

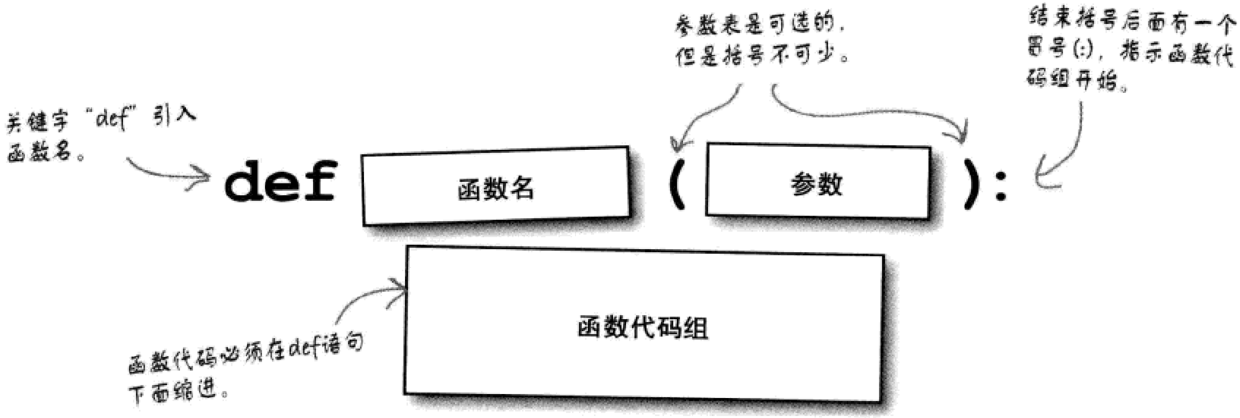
代码以这种方式重复时，大多数程序员都会寻找一种方法得到代码的一般模式，把它变成一个可重用的函数。Python程序员也不例外。通过创建一个可重用的函数，你可以根据需要调用这个函数，而不是剪切和粘贴现有的代码。

现在就来将重复代码变成一个函数。

在Python中创建一个函数

Python中的函数是一个命名的代码组，如果需要，还可以有一个参数表（可选）。

要用**def**语句定义Python函数，为函数提供一个函数名，并在括号里指定一个参数表，参数表也可以为空。以下是定义函数的标准形式：



你的函数需要做什么？

你的函数需要得到一个列表，处理列表中的各个列表项。如果它在第一个列表中发现一个嵌套列表，函数就需要重复。这可以通过在嵌套列表上调用自身来做到。换句话说，函数需要反复调用。也就是说，要在函数代码组内调用自身。

Sharpen your pencil

将所创建的函数命名为print_lol()。它有一个参数，即将在屏幕上显示的列表。拿出你的笔来，完成下面的代码，来提供所需的功能：

```
def print_lol(the_list):

    for ..............................................
        if ..............................................
            ..............................................
        else:
            ..............................................
```

Sharpen your pencil Solution

将所创建的函数命名为print _ lol()。它有一个参数，即将在屏幕上显示的列表。拿出你的笔来，完成下面的代码，来提供所需的功能：

```
def print_lol(the_list):

    for each_item in the_list:

        if isinstance(each_item, list):

            print_lol(each_item)

        else:

            print(each_item)
```

用一个"for"循环处理所提供的列表。

如果所处理的列表项本身是一个列表，则调用函数。

如果所处理的列表项不是一个列表，则在屏幕上显示这个列表项。

An IDLE Session

下面最后一次使用IDLE来测试这个新函数。它能像前面的代码一样正常工作吗？

```
>>> def print _ lol(the _ list):
        for each _ item in the _ list:
            if isinstance(each _ item, list):
                print _ lol(each _ item)
            else:
                print(each _ item)


>>> print _ lol(movies)

The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
Michael Palin
John Cleese
Terry Gilliam
Eric Idle
Terry Jones
```

定义函数。

调用函数。

同样成功了！这个递归函数确实得到了与先前代码完全相同的结果。

PDG

解决之道：递归！

通过使用递归函数，可以把14行难以理解、让人头疼的杂乱代码缩减为一个只有6行代码的函数。需要修改前面的代码才能支持额外的嵌套列表（如果影迷需要更深层次的嵌套列表），与之不同，递归函数不需要任何改变就可以正确地处理任意深度的嵌套列表。

Python 3默认为递归深度不能超过100，这意味着很多个“列表中的列表中的列表中的……”另外，如果你希望嵌套更深，还可以改变这个深度上限。



不错的起点！

通过利用函数和递归，你已经解决了先前处理列表的代码中存在的复杂性问题。

通过创建print_lol()，你已经得到一个可重用的代码块，可以在你的（以及其他人的）程序中多处使用。

现在已经做好准备，可以着手使用Python了！



你的Python工具箱

你已经读完了第1章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- “BIF” —— 内置函数。
- “组（Suite）” ——Python代码块，会通过缩进来指示分组。
- “功能齐全（Batteries included）” ——这是指Python提供了快速高效地完成工作所需的大多数功能。

IDLE说明

- IDLE shell允许你在编写代码的同时试验代码。
- 调整IDLE的首选项，以适应你的工作方式。
- 要记住：使用shell时，Alt-P表示Previous（前一个），Alt-N表示Next（下一个），不过如果你使用的是Mac，则要使用Ctrl-P和Ctrl-N。

BULLET POINTS

- 从命令行或在IDLE中运行Python 3。
- 标识符是指示数据对象的名字。标识符没有“类型”，不过标识符所指示的数据对象有类型。
- `print()` BIF会在屏幕上显示一个消息。
- 列表是一个数据集合，数据项之间用逗号分隔，整个列表用中括号包围。
- 列表就像是“打了激素”的数组。
- 可以用BIF处理列表，另外列表还支持一组列表方法。
- 列表可以存放任意数据，而且数据可以是混合类型。列表还可以包含其他列表。
- 列表可以随需要伸缩。数据使用的所有内存都由Python为你管理。
- Python使用缩进将语句归组在一起。
- `len()` BIF会提供某个数据对象的长度，或者统计一个集合中的项数，如列表中的项数。
- for循环允许迭代处理一个列表，这通常比使用一个等价的while循环更方便。
- 可以利用if… else…语句在代码中完成判定。
- `isinstance()` BIF会检查一个标识符是否指示某个指定类型的数据对象。
- 使用def来定义一个定制函数。

2 共享你的代码

函数模块



可重用的代码固然不错，不过可共享的模块更棒。

通过作为Python模块共享代码，就可以向整个Python社区开放你的代码……共享总是一件好事，不是吗？在这一章中，你将学习如何创建、安装和发布你自己的可共享模块，然后把你的模块加载到Web上的Python软件共享网站，这样所有人都能受益于你的工作。在学习过程中，你还会了解一些与Python函数有关的新技巧。

太好了，所以应该分享

你向其他程序员展示了你的函数，他们都非常喜欢。



没错，这么好的函数应该与全世界共享。

Python提供了一组技术，可以很容易地实现共享，这包括模块和一些发布工具：

- **模块允许你合理组织代码来实现最优共享。**
- **发布工具允许你向全世界共享你的模块。**

下面把函数变为一个模块，然后使用发布工具让广大的Python编程社区共享这个模块。

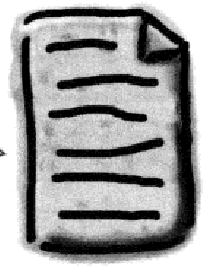
函数转换为模块

模块就是一个包含Python代码的文本文件。对模块的主要需求就是要求文件名以.py结尾，这是Python扩展名。要把你的函数转换为一个模块，需要把代码保存到一个适当命名的文件中：

第1章中的代码。

```
def print_lol(the_list):
        for each_item in the_list:
                if isinstance(each_item, list):
                        print_lol(each_item)
                else:
                        print(each_item)
```

将这个文件命名为“nester.py”。



there are no Dumb Questions

问： 最好的Python编辑器是什么？

答： 这个问题的答案取决于你问的是谁。不过，当然可以使用任何文本编辑器来创建函数的代码，并保存到一个文本文件中。像Windows的NotePad（记事本）这样简单的编辑器就能胜任这个工作，而对于Mac OS X的TextMate之类功能完备的编辑器来说更是不在话下。还有一些相当成熟的IDE，如Linux的Eclipse，以及经典的vi和emacs编辑器都可以使用。另外，你应该已经知道，Python提供了IDLE，它也包括一个内置的代码编辑器。这个编辑器可能不如那些“真正的”编辑器功能强大，不过IDLE是Python内置安装的，所以肯定可以使用。对于很多工作来说，处理Python代码时只需要IDLE的编辑窗口就完全足够了。当然，Python还有很多其他编辑器。可以查看WingIDE找到一个专门面向Python开发人员的编辑器。

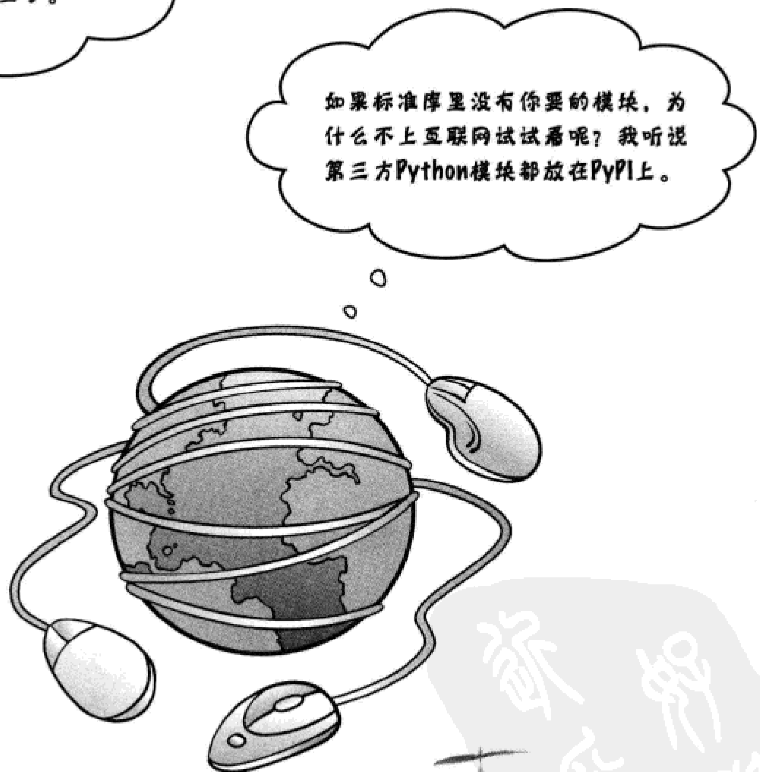
动手做！

继续创建一个名为nester.py的文本文件，其中包含第1章中的函数代码。

模块无处不在

想想，很多地方都可以找到Python模块。





PyPI读作"pie-pie"。

Python包索引（Python Package Index，PyPI）为Internet上的第三方Python模块提供了一个集中的存储库。准备好之后，就可以使用PyPI来发布你的模块，从而使你的代码可供其他人使用。你的模块已经准备就绪，不过还有一个重要的补充。

你认为你的模块还缺少点什么？

Geek Bits

你果你已经很熟悉Perl的CPAN存储库，可以认为PyPI就是与之对应的Python存储库。

注释代码

为代码加注释绝对是一个好主意。计划向全世界分享你的模块时，如果有完善的注释，这对于建立文档很有帮助。

在Python中，一个常用的注释技术是使用一个三重引号来建立多行注释。如果使用了一个三重引号，而没有将它赋至一个变量，三重引号之间的所有内容都被认为是一个注释：

你好！我是一个大块头的字符串，而且刚好也是一个Python注释。是不是很棒？

从一个三重引号开始……

```
"""This is the standard way to
include a multiple-line comment in
your code."""
```

……并以一个三重引号结束。

Sharpen your pencil

以下是你的模块代码（保存在文件nester.py中）。在下面给出的空白处写出两个注释：第一个描述模块，第二个描述函数。

把模块注释放在这里。

```
..........................................................
..........................................................
..........................................................
def print_lol(the_list):
        ..................................................
        ..................................................
        ..................................................
        for each_item in the_list:
                if isinstance(each_item, list):
                        print_lol(each_item)
                else:
                        print(each_item)
```

在这里为函数增加一个注释。

Sharpen your pencil Solution

以下是你的模块代码（保存在文件nester.py中）。在下面给出的空白处写出两个注释：第一个描述模块，第二个描述函数。

```
"""这是"nester.py"模块，提供了一个名为print_lol()的函数，这个函数的作用是打印列表，其中有可能包含（也可能不包含）嵌套列表。"""

def print_lol(the_list):
    """这个函数取一个位置参数，名为"the_list"，这可以是任何Python列表（也可以是包含嵌套列表的列表）。所指定的列表中的每个数据项会（递归地）输出到屏幕上，各数据项各占一行。"""
    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item)
        else:
            print(each_item)
```

记得加三重引号了吧？

对具体代码未做任何修改，这里只是加了一些注释。

there are no Dumb Questions

问：我怎么知道Python模块放在计算机上的什么地方？

答： 可以问IDLE。在IDLE提示窗口键入import sys; sys.path（都放在一行上），可以看到一个位置列表，Python解释器就在这些位置上搜索模块。

问：请等一下。我可以在Python程序中使用“;”把多行代码放在一行上吗？

答： 没错，这是可以的。不过，我还是建议你不要这么做。最好每个Python语句各占一行。这样不论对你还是对别人来讲，你的代码都会更易读。

问：nester.py模块放在哪里很重要吗？

答： 对现在来说没有任何影响。只要确保将它放在以后可以找到的一个地方就可以了。稍后将把这个模块安装到你的Python本地副本，这样解释器就可以找到它，而无需你记住它具体放在哪里。

问：这么说，注释就像是一个看上去很有意思的加引号的字符串，是这样吗？

答： 是的。如果一个用三重引号引起的字符串没有赋给一个变量，就会被作为一个注释。以上代码中的注释用三个双引号包围，不过也可以使用单引号。

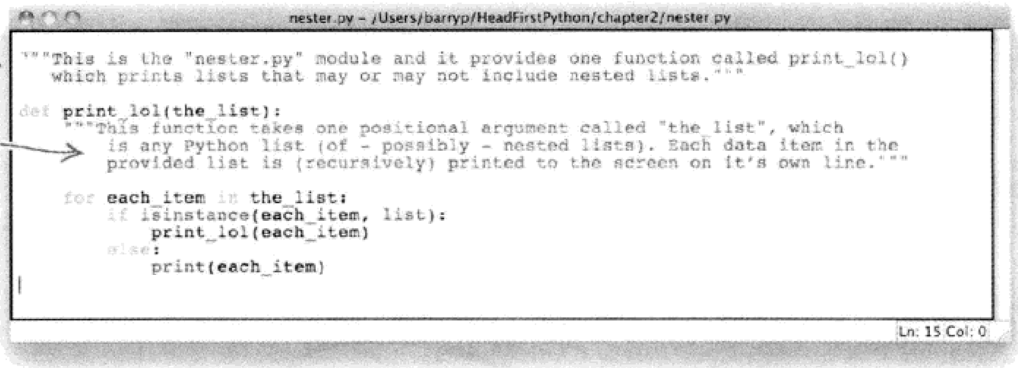
问：还有没有其他方法为Python代码加注释？

答： 有的。如果在一行中的任意位置上加了一个“#”符号，从这一点直到当前行末尾的所有内容都是注释（除非“#”出现在三重引号之间，在这种情况下，它将成为三重引号注释的一部分）。很多Python程序员测试新功能时都会利用“#”符号快速转换一行代码，使它作为注释或不作为注释。

An IDLE Session

既然已经增加了注释，而且创建了模块。下面来测试你的代码仍能正常工作。不必在IDLE提示窗口中键入函数的代码，可以在IDLE编辑窗口中加载nester.py文件，然后按F5运行这个模块的代码：

注意，注释有特定的颜色。



看起来什么也没有发生，只是Python shell“会重启”，出现一个空的提示窗口。

```
>>> ================================ RESTART ================================
>>>
>>>
```

这里确实已经发生了一些事情，Python解释器已经重置，而且执行了模块中的代码。这个代码定义了函数，但是除此以外，其他的什么也没有做。解释器只是耐心地等着你用这个新定义的函数做点什么，所以下面来创建一个列表的列表，并对这个列表调用这个函数：

```
>>> movies = [
    "The Holy Grail", 1975, "Terry Jones & Terry Gilliam", 91,
        ["Graham Chapman",
            ["Michael Palin", "John Cleese", "Terry Gilliam", "Eric Idle", "Terry Jones"]]]
```

定义第1章中电影基本信息的列表。

```
>>> print_lol(movies)
The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
Michael Palin
John Cleese
Terry Gilliam
Eric Idle
Terry Jones
```

对这个列表调用该函数。

太棒了。代码还能正常工作。这个双重列表中的数据都显示在屏幕上。

准备发布

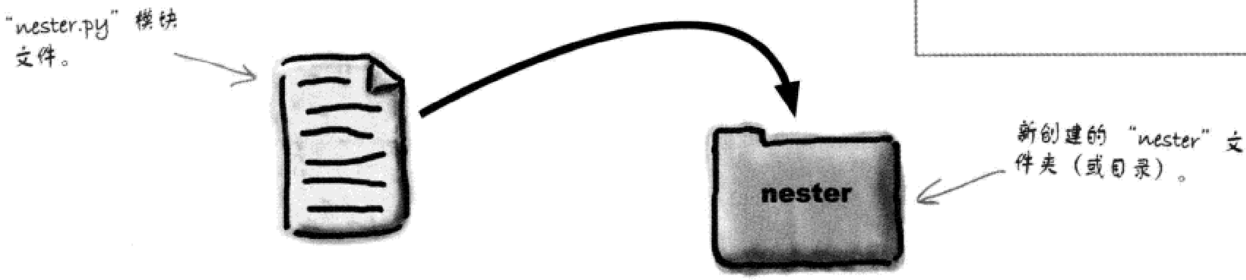
为了共享新创建的这个模块，需要准备一个发布。在Python中，所谓"发布"（distribution）是指一个文件集合，这些文件联合在一起允许你构建、打包和发布你的模块。

一旦发布存在，就可以把模块安装到你的Python本地副本上，还可以把模块上传到PyPI与全世界共享。按照下面两页给出的流程，为你的模块创建一个发布。

动手做！

按照以下几页给出的步骤实际动手。完成这些工作后，你的模块将转换为一个Python发布。

1. **首先为模块创建一个文件夹。**
   创建了文件夹之后，将`nester.py`模块文件复制到这个文件夹中。为简单起见，下面把这个文件夹命名为`nester`：



2. **在新文件夹中创建一个名为"setup.py"的文件。**
   这个文件包含有关发布的元数据。编辑这个文件，增加下面的代码：

从Python发布工具导入"setup"函数。

```
from distutils.core import setup

setup(
      name         = 'nester',
      version      = '1.0.0',
      py_modules   = ['nester'],
      author       = 'hfpython',
      author_email = 'hfpython@headfirstlabs.com',
      url          = 'http://www.headfirstlabs.com',
      description  = 'A simple printer of nested lists',
     )
```

这些是setup函数的参数名。

将模块的元数据与setup函数的参数关联。

这些只是Head First Labs对其模块使用的值，你的元数据可以与这里不同。

构建发布

现在已经有了一个文件夹，其中包含两个文件：模块代码放在nester.py中，模块的有关元数据放在setup.py中。现在来构建你的发布。

注意：如果使用Windows，需要把这些注释中的"python3"替换为"c:\Python31\python.exe"。

3 **构建一个发布文件。**

发布工具包含有构建一个发布所需的所有功能。在nester文件夹中打开一个终端窗口，键入一行命令：python3 setup.py sdist。

在提示窗口中输入这个命令。

```
File Edit Window Help Build
$ python3 setup.py sdist
running sdist
running check
warning: sdist: manifest template 'MANIFEST.in' does not exist

warning: sdist: standard file not found: should have README

writing manifest file 'MANIFEST'
creating nester-1.0.0
making hard links in nester-1.0.0......
hard linking nester.py -> nester-1.0.0
hard linking setup.py -> nester-1.0.0
creating dist
Creating tar archive
removing 'nester-1.0.0' (and everything under it)
$
```

屏幕上会出现一组状态消息，确认发布确实已经创建。

4 **将发布安装到你的Python本地副本中。**

仍然在终端窗口中，键入以下命令：sudo python3 setup.py install。

```
File Edit Window Help Install
$ sudo python3 setup.py install
running install
running build
running build_py
creating build
creating build/lib
copying nester.py -> build/lib
running install_lib
copying build/lib/nester.py -> /Library/Frameworks/Python.
framework/Versions/3.1/lib/python3.1/site-packages
byte-compiling /Library/Frameworks/Python.framework/Versions/3.1/
lib/python3.1/site-packages/nester.py to nester.pyc
running install_egg_info
Writing /Library/Frameworks/Python.framework/Versions/3.1/lib/
python3.1/site-packages/nester-1.0.0-py3.1.egg-info
```

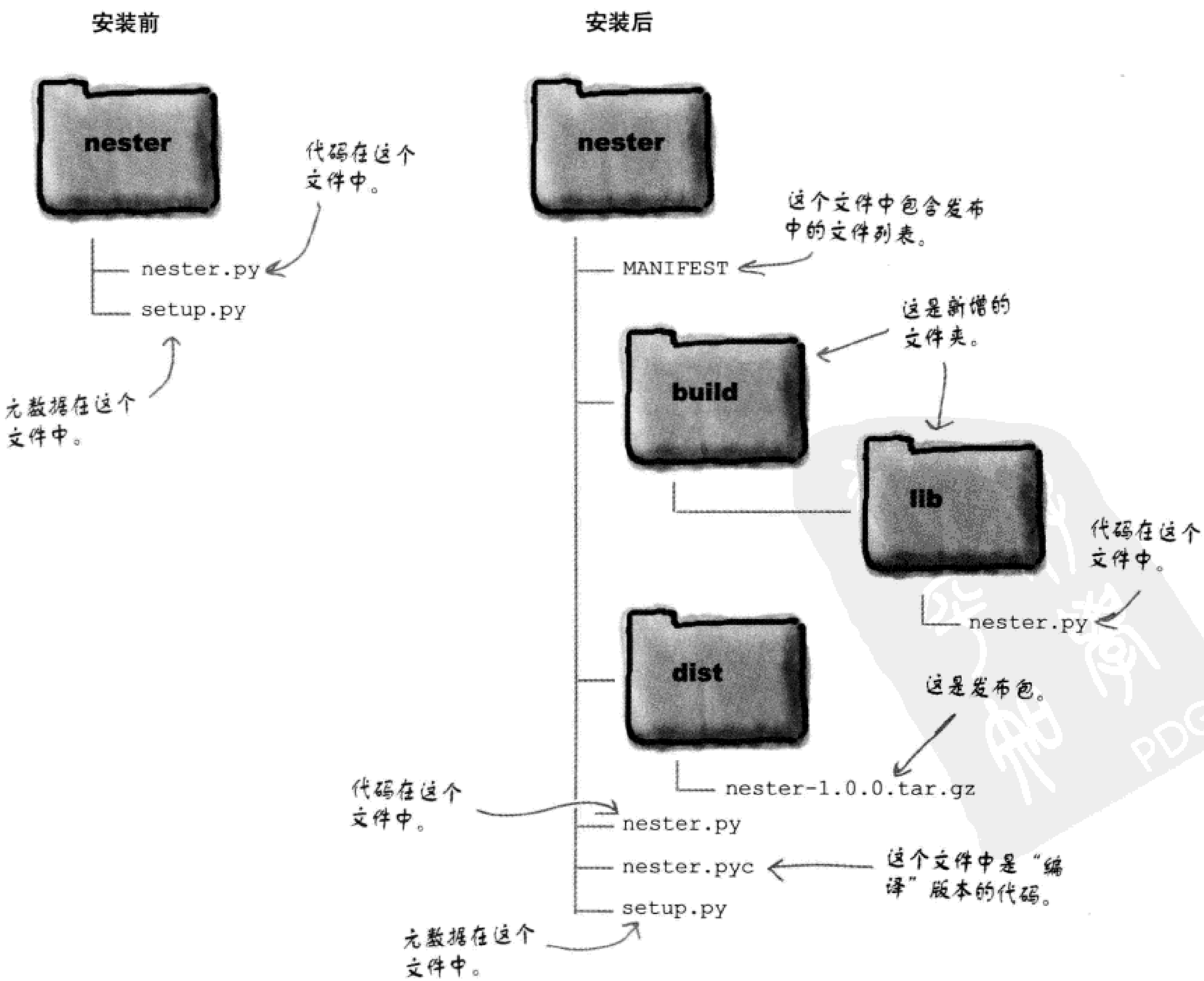
屏幕上会出现另外一组状态消息，确认发布确实已经安装。

发布已经准备就绪。

发布速览

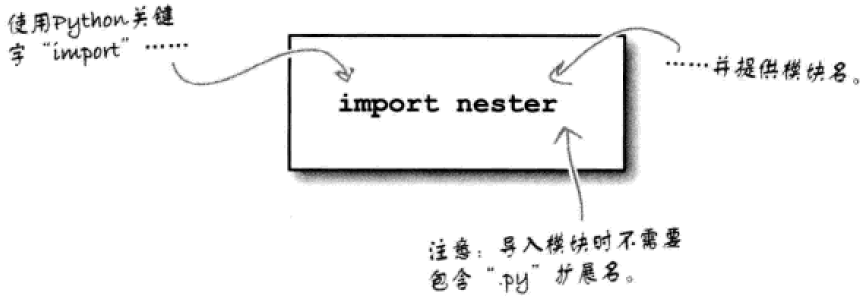
利用Python的发布工具，你的模块已经转换为一个发布，并且安装在你的Python本地副本上。

开始时只有一个函数，这个函数输入到一个名为nester.py的文件中，这就创建了一个模块。然后创建一个名为nester的文件夹存放这个模块。通过在这个文件夹中增加一个名为setup.py的文件，从而能够构建和安装你的发布，这会生成一组额外的文件，并在nester文件夹中出现两个新的文件夹。这些文件和文件夹都是由发布工具为你创建的。



导入模块并使用

既然已经构建了模块，并作为发布打包和安装，下面来看使用时还需要什么。要使用一个模块，只需把它导入到你的程序中，或者导入到IDLE shell：



import语句告诉Python将nester.py模块包含在程序中。从现在开始可以使用这个模块的函数，就好像这些函数直接输入到程序中一样，是这样吗？嗯……这可能是你所期望的。下面来看你的想法对不对。

Exercise

写一个小程序，导入你新创建的模块，并定义一个小列表，名为"cast"，然后使用模块提供的函数在屏幕上显示这个列表的内容。使用以下列表数据（都是字符串）：Palin、Cleese、Idle、Jones、Gilliam和Chapman。

....

....

....

....

....

....

在IDLE的编辑窗口中打开你的程序，然后按下F5执行代码。在下面空白处写出会发生什么：

....

....

....

Exercise Solution

写一个小程序，导入你新创建的模块，并定义一个小列表，名为“cast”，然后使用模块提供的函数在屏幕上显示这个列表的内容。使用以下列表数据（都是字符串）：Palin、Cleese、Idle、Jones、Gilliam和Chapman。

```
import nester

cast = ['Palin', 'Cleese', 'Idle', 'Jones', 'Gilliam', 'Chapman']
print_lol(cast)
```

这是一个简单的程序，只有3行代码。这里毫无困难。

在IDLE的编辑窗口中打开你的程序，然后按下F5执行代码。在下面空白处写出会发生什么：

不过这个程序无法运行！ → IDLE给出一个错误，程序无法运行！

An IDLE Session

在IDLE编辑窗口中输入你的程序，按下F5（或者从Run菜单选择Run Module），确实会出现问题：

try_nester.py - /Users/barryp/HeadFirstPython/chapter2/try_nester.py

```
import nester

cast = ['Palin', 'Cleese', 'Idle', 'Jones', 'Gilliam', 'Chapman']
print_lol(cast)
```

Ln: 6 Col: 0

看起来程序没有执行，而会报告一个错误消息：

```
>>> ================================ RESTART ================================
>>>
Traceback (most recent call last):
  File "/Users/barryp/HeadFirstPython/chapter2/try_nester.py", line 4, in <module>
    print_lol(cast)
NameError: name 'print_lol' is not defined
>>>
```

在IDLE中输入程序，按下F5时产生一个NameError错误……看起来没有找到你的函数！！！

Python的模块实现命名空间

Python中的所有代码都与一个命名空间关联。

主Python程序中（以及IDLE shell中）的代码与一个名为__main__的命名空间关联。将代码放在其单独的模块中时，Python会自动创建一个与模块同名的命名空间。所以，你的模块中的代码会与一个名为nester的命名空间关联。

在“main”前面和后面分别有两个下划线。



没错，命名空间名就像是姓氏。

假设你目前正在某个命名空间中，如果想指示另外一个模块命名空间中的某个函数，就需要用该模块的命名空间名对这个函数的调用做出限定。

所以，不能简单地通过`print_lol(cast)`调用这个函数，而需要限定为`nester.print_lol(cast)`。这样一来，Python解释器就会知道去哪里找这个函数。命名空间限定的格式如下：首先是模块名，然后是一个点号，再后面是函数名。

模块名，用来标识命名空间。

```
nester.print_lol(cast)
```

然后正常调用函数，提供“cast”作为要处理的列表。

点号将模块命名空间与函数名分隔开。

PDG

An IDLE Session

下面来做个测试。仍然在IDLE shell中，导入你的模块，创建列表，再尝试调用这个函数而不提供限定名。你会看到一个错误消息：

```
>>> import nester
>>> cast = ['Palin', 'Cleese', 'Idle', 'Jones', 'Gilliam', 'Chapman']
>>> print_lol(cast)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print_lol(cast)
NameError: name 'print_lol' is not defined
```

不出所料，你的代码会导致一个NameError错误，因为你没有对名做出限定。

用命名空间限定函数名时，情况大为好转：

```
>>> nester.print_lol(cast)
Palin
Cleese
Idle
Jones
Gilliam
Chapman
```

这一次程序能正常工作了……列表中的数据项分别显示在屏幕上。

Geek Bits

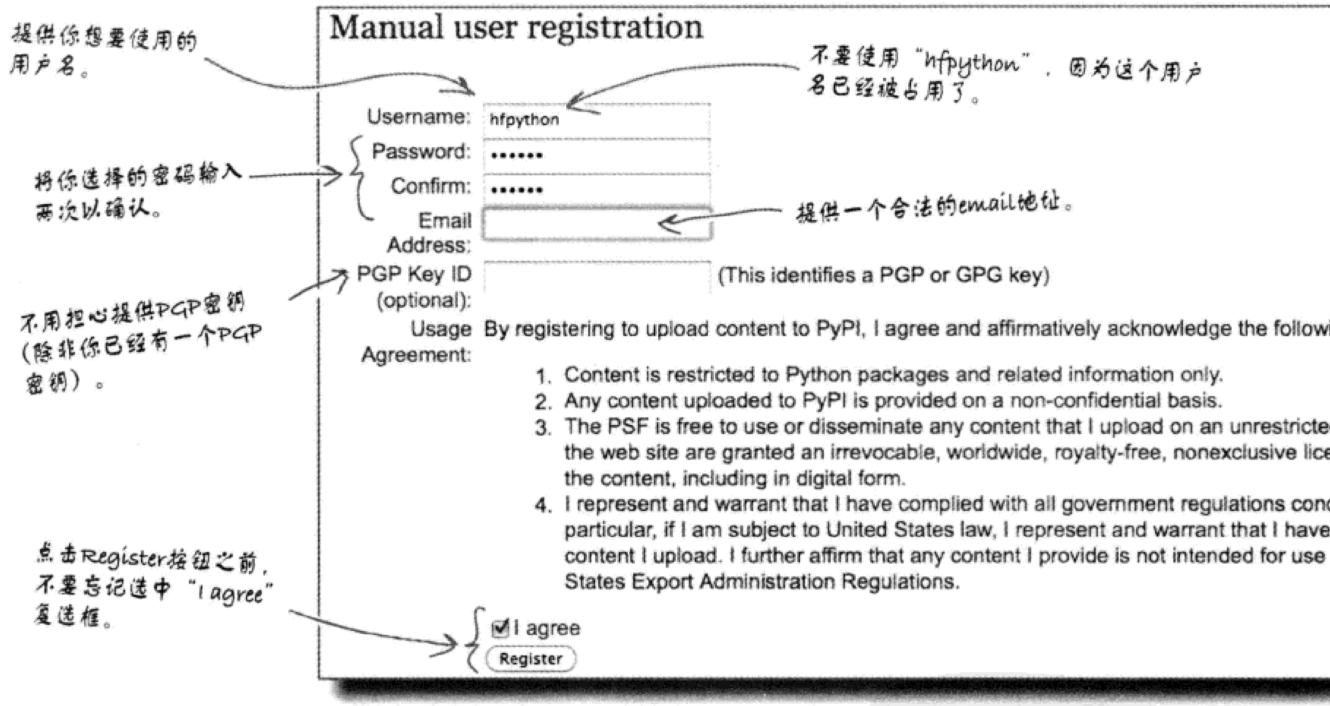
使用一个普通的import语句时，如`import nester`，这会指示Python解释器允许你使用命名空间限定来访问nester的函数。不过，还可以更为特定。如果使用`from nester import print_lol`，会把指定的函数（这里就是`print_lol`）增加到当前命名空间中，这样一来，就不必再使用命名空间限定。不过需要注意，如果你的当前命名空间中已经定义了一个名为`print_lol`的函数，这个特定`import`语句会用导入的函数覆盖你自己定义的函数，而这可能并不是你原来期望的行为。

现在可以把你的模块上传到PyPI了。

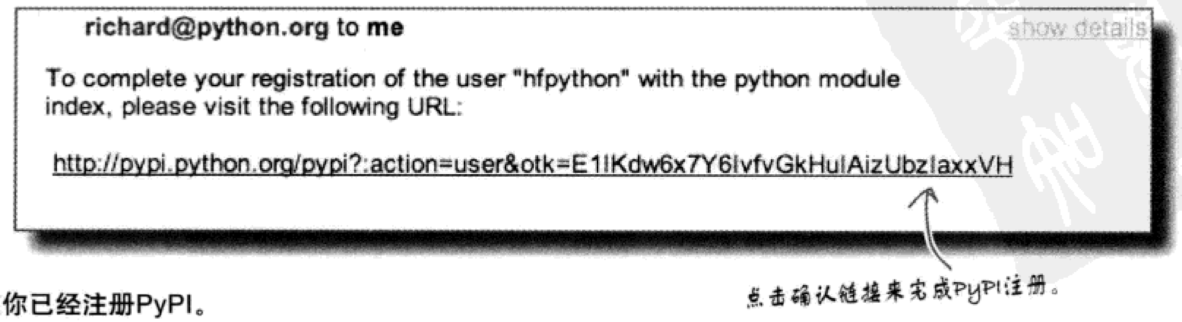
注册PyPI网站

为了向PyPI上传你的发布，需要在PyPI网站注册。这个过程非常简单。

首先访问 PyPI网站（*http://pypi.python.org/*），并请求一个PyPI ID：



如果所有注册信息都没问题，会向你在注册表单上提交的email地址发出一个确认消息。这个email消息包含一个链接，可以点击这个链接来确认你的PyPI注册：



现在你已经注册PyPI。

向PyPI上传代码

一切准备就绪！已经把函数中的代码放在一个模块中，用来创建一个发布，而且已经安装在你的Python本地副本中。要把这个发布上传到PyPI，需要完成以下两个步骤：通过命令行窗口注册PyPI，以及通过命令行窗口上传。

这里需要再次注册PyPI，这看起来让人有些奇怪，因为我们刚在PyPI网站注册过。不过，命令行上传工具需要知道你的PyPI用户名和密码，这正是这个注册所要做的。不用担心：这个工作只需要做一次。

File Edit Window Help Register

```
$ python3 setup.py register
running register
running check
We need to know who you are, so please choose either:
 1. use your existing login,
 2. register as a new user,
 3. have the server generate a new password for you (and email it to you), or
 4. quit
Your selection [default 1]:
1
Username: hfpython
Password:
Registering nester to http://pypi.python.org/pypi
Server response (200): OK
I can store your PyPI login so future submissions will be faster.
(the login will be stored in /Users/barryp/.pypirc)
Save your login (y/N)?y
```

指示要注册你的详细信息。

确认你想使用刚创建的PyPI凭据（用户名和密码）。

使用你的PyPI设置并保存，以备以后使用。

输入并保存注册详细信息后，现在可以向PyPI上传你的发布了。需要用另一个命令行来完成这个工作：

注意：如果试图上传一个名为"nester"的模块，你会得到一个错误，指出这个名字已经被占用。:-)

File Edit Window Help Upload

```
$ python3 setup.py sdist upload
running sdist
running check
reading manifest file 'MANIFEST'
creating nester-1.0.0
making hard links in nester-1.0.0.....
hard linking nester.py -> nester-1.0.0
hard linking setup.py -> nester-1.0.0
Creating tar archive
removing 'nester-1.0.0' (and everything under it)
running upload
Submitting dist/nester-1.0.0.tar.gz to http://pypi.python.org/pypi
Server response (200): OK
$
```

指示向PyPI上传你的软件发布。

确认上传成功。你的发布现在已经成为PyPI的一部分。

欢迎来到PyPI社区

祝贺你！你现在已经是PyPI社区中一个合格的正式成员了。你的发布已经加入到PyPI中，还有超过10000个其他上传软件与之为伍。可以访问PyPI网站来确认你的发布确实已经上传。

世界各地的程序员现在都能下载和解压你的模块，并安装到他们的Python本地副本中，想到这一点肯定会让你感觉很好。

坐下来，安心等着大家的喝采吧…… ← 现在你已经编写并发布了代码……太酷了！

there are no Dumb Questions

问：哪一个更好一点：是普通的import还是特定的import？

答： 实际上，并不能说它们哪一个更好。大多数程序员都会根据自己的个人喜好和风格混合使用这两种导入方式（不过确实有很多程序员争相认为他们首选的方法才是“真正的好方法”）。

需要指出，“from module import function”形式会“搅乱”你的当前命名空间，因为当前命名空间中已定义的名字会被导入的名字重写。

问：在IDLE编辑窗口中按下F5时，就好像用一个import语句导入了这个模块的代码一样，是这样吗？

答： 是的，确实是这样。编辑窗口中的代码会由Python编译和执行，编辑窗口的所有名字都会导入到IDLE shell所用的命名空间。这很方便，因为这样一来，可以在IDLE中很容易地测试功能。不过要记住，要想在IDLE之外使用模块的功能，仍然需要先导入模块。

问：把模块安装到我的Python本地副本中确实有必要吗？能不能直接放在某个原有的文件夹中再从那里导入？

答： 嗯，这是可以的。不过要记住，Python会在一组特定的位置寻找模块（应该还记得本章前面介绍的import sys; sys.path技巧吧）。如果把你的模块放在某个文件夹里，而这个文件夹并未列在Python的路径列表中，解释器就可能无法找到你的模块，这就会导致ImportError错误。通过使用发布工具来构建模块并安装到你的Python本地副本中，就能避免这种错误。

问：我注意到发布工具创建了一个名为nester.pyc的文件？这到底是什么？

答： 这个问题问得好。解释器第一次执行模块代码时，它会读入代码，并转换为一种中间字节码格式，最终所要执行的就是这些字节码（这种思想与Java JVM的工作原理很类似：利用Java技术编译代码时，Java代码会转换为一个类文件）。Python解释器非常聪明，下一次使用模块时它会跳过这个转换阶段，因为它能确定原来的模块代码文件何时有变化。如果你的模块代码没有改变，就不会发生任何转换，相应地将会执行“编译”版本的代码。如果代码确实有改变，则会发生转换（创建一个新的pyc文件）。所有这些工作的好处是：Python看到一个pyc文件时，它会尝试使用这个文件，因为这样做可以让程序运行快得多。

问：太棒了。这么说，我只向用户提供pyc文件就可以了？

答： 不行，绝对不能这样做，因为pyc文件（如果能找到）的使用主要是解释器完成的一个运行时优化。

问：那么，如果我不需要pyc文件，能不能把它删除？

答： 当然可以，如果你确实想这么做的话。不过要记住，你可能会损失可能的运行时优化。

成功的代价是责任

有很多来自不同地方的程序员在使用你的模块。其中有些程序员还需要更多的特性。



不可避免需要做出改变

你要让当前的用户满意，维护现有的功能，同时还要为有需求的那些用户提供改进的功能。这可能很棘手。

你有哪些选择？

生活中处处有选择

要确定这里该怎么做，可能有很多建议。

这太容易了。只需要再创建一个函数，比如名为“print_lol2”，对不对？然后可以使用特定形式的import语句导入你想用的函数。这真的并没有那么难办……



对，看起来这是可行的。

你可以编辑模块的代码，定义一个名为`print_lol2`的新函数，然后编写这个函数来完成嵌套打印。想用原函数时，可以使用这个特定形式的import语句：`from nester import print_lol`。如果想用新函数（即改进版本的函数），则使用以下`import`语句：`from nester import print_lol2`。

这是可以的，不过……

但是这个建议会让工作加倍……有时这可能是可以的……但是创建第二个函数（而且这个函数与前一个几乎完全相同）在我看来很浪费。



没错。定义第二个函数确实很浪费。

这样做不仅向模块引入了一个几乎相同的函数，这有可能导致维护恶梦，而且还会让模块的用户使用模块时困难得多，他们必须提前决定需要哪一个版本的函数。增加第二个函数会让模块的应用编程接口（Application Programming Interface，API）变得不必要的过于复杂。

肯定还有更好的策略，是不是？

用额外的参数控制行为

如果向函数增加一个额外的参数，不用太麻烦就可以用现在的代码处理缩进。



让你的函数更上一层楼

目前，你的函数只有一个参数：`the_list`。如果再增加第二个参数`level`，就可以用它来控制缩进。如果`level`为正值，这指示在屏幕上显示一行数据时需要加多少个制表符（tab）。如果`level`为0，则不使用缩进；如果值为1，只使用一个制表符；如果为2，就使用2个制表符；依此类推。

很显然，这里需要有一种循环机制，对不对？你已经知道如何对一个大小可变的列表进行迭代，那么在Python中如何迭代固定次数呢？

Python有没有相应的功能来提供帮助？

写新代码之前，先考虑BIF

如果遇到一个需求，而且你认为这个需求很普遍，先问问自己有没有一个内置函数（BIF）可以提供帮助。毕竟，迭代固定次数是一个经常要做的事情。

另外要记住：Python 3包含有70多个BIF，所以有大量现成的功能等着你来发现。

WHO DOES WHAT?

将各个BIF与正确的描述连线。我们已经为你完成了第一个连线。所有线都连好后，圈出你认为这个函数的下一版本中要用的BIF。

BIF	BIF的作用
`list()`	创建成对数据的一个编号列表，从0开始。
`range()`	返回一个Python数据对象的唯一标识。
`enumerate()`	这是一个工厂函数，创建一个新的空列表。
`int()`	返回一个可迭代数据结构（如列表）中的下一项。
`id()`	返回一个迭代器，根据需要生成一个指定范围的数字。
`next()`	将一个字符串或另一个数转换为一个整数（如果可行）。

（图中已画出一条连线：`list()` → “这是一个工厂函数，创建一个新的空列表。”）

WHO DOES WHAT? 答案

将各个BIF与正确的描述连线。我们已经为你完成了第一个连线。所有线都连好后，圈出你认为这个函数的下一版本中要用的BIF。

BIF

- list()
- range()
- enumerate()
- int()
- id()
- next()

这个BIF看起来很有意思。

BIF的作用

- 创建成对数据的一个编号列表，从0开始。
- 返回一个Python数据对象的惟一标识。
- 这是一个工厂函数，创建一个新的空列表。
- 返回一个可迭代数据结构（如列表）中的下一项。
- 返回一个迭代器，根据需要生成一个指定范围的数字。
- 将一个字符串或另一个数转换为一个整数（如果可行）。

range() BIF迭代固定次数

range() BIF可以提供你需要的控制来迭代指定的次数，而且可以用来生成一个从0直到（但不包含）某个数的数字列表。以下是这个BIF的用法：

```
for num in range(4):
    print(num)
```

"num"是目标标识符，会逐个赋值为"range()"生成的各个数字。

生成直到（但不包括）4的数字。

数字0、1、2和3会出现在屏幕上。

there are no Dumb Questions

问： **要在我的程序中使用BIF，真的不需要导入这些BIF吗？**

答： 不需要。总之，这些BIF会专门导入到每一个Python程序和IDLE中。

问： **这么说，这些BIF肯定属于__main__命名空间，是吗？**

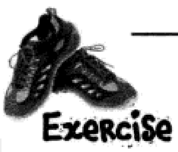
答： 并非如此。它们会自动导入到`__main__`命名空间，但是这些BIF有它们自己的命名空间，这个命名空间名为`__builtins__`（后面将会介绍）。

问： **我知道range()怎么用了，不过直接用一个while循环也能很容易地完成同样的工作，不是吗？**

答： 没错，确实如此，不过这种做法没有使用`range()`这么优雅。说实在的，如果使用while循环，不仅需要写更多的代码，还要求你负责考虑循环状态，而`range()`会为你考虑这些问题。一般来讲，Python程序员会想方设法减少需要编写和操心的代码，这样可以得到更好的代码健壮性，错误更少，当然还能让程序员睡个好觉。

问： **这么说，BIF确实对我有好处，是吗？**

答： BIF的存在就是希望通过提供一个函数集让你的编程过程尽可能简单直接，这些函数可以为常见的问题提供常用的解决方案。由于它们包含在Python中，所以你可以相信这些函数已经得到充分的测试，可以兑现它的“诺言”。你可以完全信赖这些BIF。使用这些BIF能让你如虎添翼，倍感轻松。所以，没错，BIF确实对你很有好处！



既然对`range()` BIF已经有所认识，下面修改你的函数来使用`range()`，让嵌套列表缩进指定数目的制表符。

提示：要使用`print()` BIF在屏幕上显示一个制表符（TAB），而不是简单换行（这是`print()`的默认行为），需要使用以下Python代码：`print("\t", end='')`。

```
"""这是"nester.py"模块，提供了一个名为print_lol()的函数
   用来打印列表，其中包含或不包含嵌套列表。"""

def print_lol(the_list, ...............................):
    """这个函数有一个位置参数，名为"the_list",
       这可以是任何Python列表（包含或不包含嵌套列表），
       所提供列表中的各个数据项会（递归地）打印到屏幕上，而且各占一行。"""
       ..................................................................
    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item)
        else:
            ...............................................
                  ..................................................
            print(each_item)
```

加入另一个参数名。

不要忘记编辑注释。

在这里增加代码来加入所需数目的制表符。

Exercise Solution

既然对range() BIF已经有所认识，下面修改你的函数来使用range()，让嵌套列表缩进指定数目的制表符。

提示：要使用print() BIF在屏幕上显示一个制表符（TAB），而不是简单换行（这是print()的默认行为），需要使用以下Python代码：print("\t", end='')。

```
"""这是"nester.py"模块，提供了一个名为print_lol()的函数
   用来打印列表，其中包含或不包含嵌套列表。"""

def print_lol(the_list, level):
    """这个函数有一个位置参数，名为"the_list"，
       这可以是任何Python列表（包含或不包含嵌套列表），
       所提供列表中的各个数据项会（递归地）打印到屏幕上，而且各占一行。
       第二个参数（名为"level"）用来在遇到嵌套列表时插入制表符。"""
    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item)
        else:
            for tab_stop in range(level):
                print("\t", end='')
            print(each_item)
```

使用"level"的值来控制使用多少个制表符。

每一层缩进显示一个TAB制表符。

An IDLE Session

现在来测试这个函数的新版本。将你的模块文件加载到IDLE，按下F5将函数导入到IDLE的命名空间，然后对电影列表调用这个函数并提供第二个参数：

```
>>> print_lol(movies, 0)
The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print_lol(movies,0)
  File "/Users/barryp/HeadFirstPython/chapter2/nester/nester.py", line 14, in print_lol
    print_lol(each_item)
TypeError: print_lol() takes exactly 2 positional arguments (1 given)
```

调用函数，一定要提供第二个参数。

"movies"中的数据开始出现在屏幕上……

……然后，真倒霉，代码崩溃了！这里有问题。

这里给出了线索，从中可以看出哪里出了问题。

你的代码存在一个TypeError错误，正是这个错误导致代码崩溃。

Python会尽力运行你的代码

与编译型语言不同（如C、Java、C#以及其他语言），Python在运行之前并不全面检查代码的合法性。这就使得Python可以做很多在其他语言中无法做到的很酷的事情，如在运行时动态地定义函数。当然，这相当灵活，也非常强大。

不过，与此同时也是有代价的，编写代码时你必须非常小心，因为通常能够被一个传统静态类型的编译型语言捕获和标志的“错误”在Python中则很可能顺利溜过。





BRAIN POWER

再来看上一页上的错误。你认为Python为什么会给出这个错误？你觉得哪里有问题？

跟踪代码

想要找出看似正确的程序中哪里出了问题，可以使用一个很有用的技术，这就是跟踪各行代码执行时到底发生了什么。下面是你现在使用的代码。只有3行代码（记住：列表的创建只是一行代码），看起来不应该有任何麻烦：

```
import nester

movies = [ "The Holy Grail", 1975, "Terry Jones & Terry Gilliam",
           91,["Graham Chapman", ["Michael Palin",
           "John Cleese", "Terry Gilliam", "Eric Idle",
           "Terry Jones"]]]

nester.print_lol(movies, 0)
```

这两行看起来没问题。

你在调用函数，并提供了两个参数，所以这也没问题。

数据赋至函数的参数后，开始在所传入列表中包含的各个数据项上执行这个函数的代码：

```
def print_lol(the_list, level):
    """This function ...... """

    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item)
        else:
            for tab_stop in range(level):
                print("\t", end='')
            print(each_item)
```

"movies"列表赋给"the_list"另外"level"赋值为0。

为了节省空间，这里没有给出完整的注释。

处理列表中的各项……

……然后根据数据项是否是一个列表来确定下一步做什么。

如果数据项确实是一个列表，递归地调用这个函数……请等等，这看起来好像不太对劲！

找出哪里出了问题

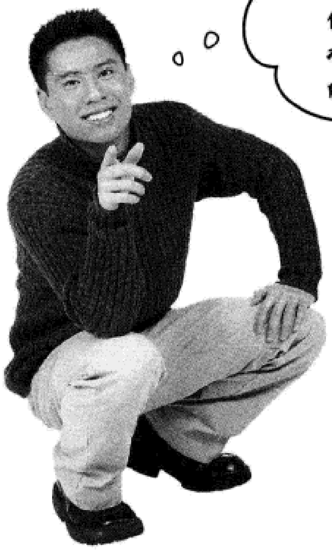
你的问题在于：函数的递归调用仍然在使用原来的函数签名（只需要一个参数）。函数的新版本要求有两个参数。

修正这个问题很容易：只需在调用函数的新版本时提供数目正确的参数。所以函数中的以下代码：

```
if isinstance(each_item, list):
    print_lol(each_item)
```

需要重写，指定的参数个数要正确：

```
if isinstance(each_item, list):
    print_lol(each_item, level)
```



没错。“level”的使用确实还有问题，需要做最后一点调整。

增加level参数的目的就是为了能控制嵌套输出。每次处理一个嵌套列表时，都需要将level的值增1。你的代码段应该是这样的：

```
if isinstance(each_item, list):
    print_lol(each_item, level+1)
```

只需在每次递归调用函数时将level值增1。

现在来完成这个更新。

用你的新代码更新PyPI

继续编辑nester.py模块（在nester文件夹中），让它正确地调用你的函数。既然已经有了一个新版本的模块，最好更新之前上传到PyPI的发布。

修改代码后，还需要对发布的setup.py程序做一个小小的修改。你已经修改了API，所以需要调整setup.py中与version关联的值。下面从版本1.0.0前进到1.1.0：

```
from distutils.core import setup

setup(
        name         = 'nester',
        version      = '1.1.0',
        py_modules   = ['nester'],
        author       = 'hfpython',
        author_email = 'hfpython@headfirstlabs.com',
        url          = 'http://www.headfirstlabs.com',
        description  = 'A simple printer of nested lists',
     )
```

修改与"version"关联的值，指示PyPI这实际上是一个新版本。

就像创建和上传发布时所做的一样，在发布文件夹中调用setup.py程序来完成上传：

```
File Edit Window Help UploadAgain
$ python3 setup.py sdist upload
running sdist
running check
reading manifest file 'MANIFEST'
creating nester-1.1.0
making hard links in nester-1.1.0.....
hard linking nester.py -> nester-1.1.0
hard linking setup.py -> nester-1.1.0
Creating tar archive
removing 'nester-1.1.0' (and everything under it)
running upload
Submitting dist/nester-1.1.0.tar.gz to http://pypi.python.org/pypi
Server response (200): OK
$
```

你肯定喜欢这些"200 OK"消息，对不对？

你的新发布现在已经在PyPI上了。



Mark：大家来看看这个……PyPI上已经更新了nester模块。

Bob：版本1.1.0……

Laura：我想知道有什么变化？

Mark：它仍然能处理嵌套的列表，不过现在还可以在屏幕上看到嵌套的结构，我觉得这实在太酷了。

Laura：而且很有用。我一直盼望着有这样一个特性。

Bob：嗯……是的……不过怎么升级我现在的本地副本呢？

Mark：只需要像第一次从PyPI下载和安装nester时那样，按同样的步骤做就行了。

Bob：这么说，我要下载包文件，解压缩，让setup.py为我把它安装到我的Python中，是吗？

Mark：是的，再没有比这更容易的了。

Laura：那么nester的现在这个版本呢？这个“老”版本会怎么样？

Bob：对呀……我现在有两个nester模块吗？

Mark：不是的。使用setup.py安装最新版本时，它会变成当前版本，完全取代原来的模块（也就是1.0.0版本）。

Bob：PyPI也知道要提供这个模块的最新版本，是吧？

Mark：没错，你访问PyPI网站搜索nester时，总会向你提供这个模块的最新版本。

Laura：哦，我经常使用这个模块，而且一直在盼望这个特性。我想我现在就要更新。

Mark：我已经把我的模块升级了，这个新模块表现不错。

Bob：对，我也经常用这个模块，所以我想该更新我的系统，安装这个最新版本。毕竟依赖一个过时的软件总不是个好主意，对不对？

Mark：我也这么认为。而且计划总不如变化快。

Laura：以后再聊。我还有些事要做。

Bob：我也是。我要到PyPI下载最新的nester，把它安装到我的Python本地副本中。我还要快速测试一下，确保一切正常。

Mark：朋友们，回头见……

你改变了API

你的新版本nester确实更好一些，但是并不是所有用户都这么认为。



由于你匆匆发布模块最新最棒的版本，以至于忽略了现有的一些用户。应该记得，并不是所有用户都希望有这个新的嵌套打印特性。不过，由于向print_lol()增加了第二个参数，你已经改变了函数的签名，这意味着你的模块有一个不同的API。使用原来的API的人就会遇到问题。

理想的解决方案是同时提供这两个API，一个API提供新特性，而另一个不提供。也许这个特性可以是可选的？

但是怎么做到呢？

使用可选参数

为了将一个函数的必要参数变成可选的参数，需要为这个参数提供一个缺省值。如果没有提供参数值，就会使用这个缺省值。如果提供了一个参数值，则会使用这个值而不是缺省值。当然，关键在于，参数的缺省值实际上使得这个参数成为可选参数。

为了向函数参数提供一个缺省值，需要在参数名后面指定这个缺省值：

这两个参数都是必要的。

```
def print_lol(the_list, level):
```

```
def print_lol(the_list, level=0):
```

增加一个缺省值使得"level"变成一个可选的参数。

定义了参数的缺省值之后，现在可以采用多种不同方式调用这个函数：

```
nester.print_lol(movies, 0)

nester.print_lol(movies)

nester.print_lol(movies, 2)
```

调用函数，并提供两个参数。

调用函数，提供一个参数，第二个参数使用缺省值。

调用函数，并提供两个参数，不过为第二个参数提供另一个起始值。

PDG

你的函数现在支持不同的签名，不过功能还照旧。

An IDLE Session

修改你的代码，为`level`参数给定一个缺省值0，然后将你的代码加载到IDLE编辑器。按下F5将代码加载到shell，进一步确认函数的这个最新版本能够正常工作。首先定义一个简短的双重列表，使用这个函数在屏幕上显示这个列表：

```
>>> names = ['John', 'Eric', ['Cleese', 'Idle'], 'Michael', ['Palin']]
>>> print_lol(names, 0)
John
Eric
        Cleese
        Idle
Michael
        Palin
```

正如所料，这里提供了标准行为，将嵌套列表缩进。

现在尝试做同样的事情，不过不再指定第二个参数，我们要依靠它的缺省值：

```
>>> print_lol(names)
John
Eric
        Cleese
        Idle
Michael
        Palin
```

没有指定第二个参数，这里使用缺省值，也能正常工作。

现在为第二个参数指定一个值，注意函数行为有什么变化：

```
>>> print_lol(names, 2)
                John
                Eric
                        Cleese
                        Idle
                Michael
                        Palin
```

为第二个参数指定另一个值，缩进将从指定的级别开始。

最后这个例子中为第二个参数提供了一个看上去很像的值。看看会发生什么：

```
>>> print_lol(names, -9)
John
Eric
Cleese
Idle
Michael
Palin
```

如果使用一个负值，实际上这会“关闭”缩进，因为“level”数不可能是一个负整数。这与版本1.0.0原来的输出非常相似，是不是？

模块支持两个API

漂亮！看起来你的模块很不错，因为两个API（原来的1.0.0 API和新的1.1.0 API）现在都能使用。

下面花点时间来创建并向PyPI上传一个新的发布。像前面一样，先修改setup.py程序中的version设置：

```
name        = 'nester',
version     = '1.2.0',
py_modules  = ['nester'],
```

再次确保改变"setup.py"中与"version"关联的值。

修改代码后，将这个新版本的发布上传到PyPI：

```
File  Edit  Window  Help  UploadThree
$ python3 setup.py sdist upload
running sdist
running check
reading manifest file 'MANIFEST'
creating nester-1.2.0
making hard links in nester-1.2.0......
hard linking nester.py -> nester-1.2.0
hard linking setup.py -> nester-1.2.0
Creating tar archive
removing 'nester-1.2.0' (and everything under it)
running upload
Submitting dist/nester-1.2.0.tar.gz to http://pypi.python.org/pypi
Server response (200): OK
$
```

看起来一切正常，干得漂亮。

大功告成！setup.py给出的消息确认了你的最新版本nester已经上传到PyPI。希望这个版本能让所有用户都满意。

BRAIN POWER

仔细考虑你的代码。用户使用这个版本的代码时怎么还会有问题？

API还是不对

尽管这个API允许用户按原来的形式调用函数，但是默认情况下会打开嵌套打印。并不是所有人都需要这种行为，而且有些人根本不希望这样。



当然，如果确实希望某个功能是可选的（也就是说，不是默认的），就应该调整代码来保证这一点。不过怎么做到呢？

一种解决方案是增加第三个参数，需要缩进时就设置为True，不需要缩进时则设置为False。如果确保这个参数默认为False，原来的功能就会成为默认行为，代码的用户必须显式请求这个新的缩进特性。

下面来看如何增加最后这个修改。

Exercise

❶ 最后一次修改模块，为函数增加第三个参数。将这个参数命名为indent，开始时这个参数值设置为False——也就是说，默认情况下不打开缩进特性。在函数体中使用indent值来控制实现缩进的代码。

注意：为了节省空间，这里不再显示模块的注释。当然，需要对注释做必要的调整，使注释与代码一致。

将新加的参数放在这里。

```
def print_lol(the_list, ..................., level=0):

    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item, ..................., level+1)
        else:
            ...............................
                for tab_stop in range(level):
                    print("\t", end='')
            print(each_item)
```

这里需要放什么？

增加一行代码来控制何时缩进。

❷ 增加新代码后，在向PyPI上传模块的最新版本之前，需要如何编辑setup.py程序？请给出你的建议：

....

....

❸ 你会使用什么命令向PyPI上传你的新发布：

....

....

Exercise Solution

1. 最后一次修改模块，为函数增加第三个参数。将这个参数命名为indent，开始时这个参数值设置为False——也就是说，默认情况下不打开缩进特性。在函数体中使用indent值来控制实现缩进的代码。

包含缺省值了吗？

```
def print_lol(the_list, indent=False , level=0):

    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item, indent , level+1)
        else:
            if indent :
                for tab_stop in range(level):
                    print("\t", end='')
            print(each_item)
```

签名已经改变，所以一定要更新这个调用。

不要忘记“if”代码行末尾的冒号。

一个简单的“if”语句就可以达到目的。

这个“for”循环还有一个漂亮的候选做法，可以替换为以下代码：print（“\t” * level, end=' '）。

2. 增加新代码后，在向PyPI上传模块的最新版本之前，需要如何编辑setup.py程序？请给出你的建议：

编辑“setup.py”使得version = '1.3.0',

这是模块的一个新版本，所以要修改“setup.py”文件中与“version”关联的值。

3. 你会使用什么命令向PyPI上传你的新发布：

python3 setup.py sdisk upload

要记住：如果使用Windows，则要用“C:\Python31\python.exe”而不是“python3”。

An IDLE Session

对功能再做最后一次测试，这样能让你相信模块现在确实按你和你的用户所希望的方式工作。下面从原来的默认行为开始：

```
>>> names = ['John', 'Eric', ['Cleese', 'Idle'], 'Michael', ['Palin']]
>>> print_lol(names)
John
Eric
Cleese
Idle
Michael
Palin
```

恢复原来的默认功能（这会让Bob很高兴）。

接下来，提供True作为第二个参数来打开缩进：

```
>>> names = ['John', 'Eric', ['Cleese', 'Idle'], 'Michael', ['Palin']]
>>> print_lol(names, True)
John
Eric
	Cleese
	Idle
Michael
	Palin
```

通过提供第二个参数，可以打开缩进输出（这会让Laura很满意）。

最后，通过提供第三个参数值控制缩进从哪里开始：

```
>>> names = ['John', 'Eric', ['Cleese', 'Idle'], 'Michael', ['Palin']]
>>> print_lol(names, True, 4)
				John
				Eric
					Cleese
					Idle
				Michael
					Palin
```

还可以从一个指定的制表符缩进。

动手做！ → 编辑你的setup.py文件，然后把发布上传到PyPI。

模块重获声誉

祝贺你！改进后的新模块很快赢得了很好的口碑。

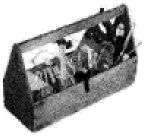


你的Python技艺已初见雏形

你已经创建了一个有用的模块，使它能够共享，而且上传到PyPI网站。全世界的程序员都可以下载这个模块，并在他们的项目中使用你的代码。

继续努力。

PDG



你的Python工具箱

你已经读完了第2章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- 使用“三重引号字符串”可以在代码中加入一个多行注释。
- “PyPI”就是Python包索引（Python Package Index），这个网站很值得访问。
- Python内存中的名字就存放在“命名空间”中。
- Python的主命名空间名为__main__。

IDLE说明

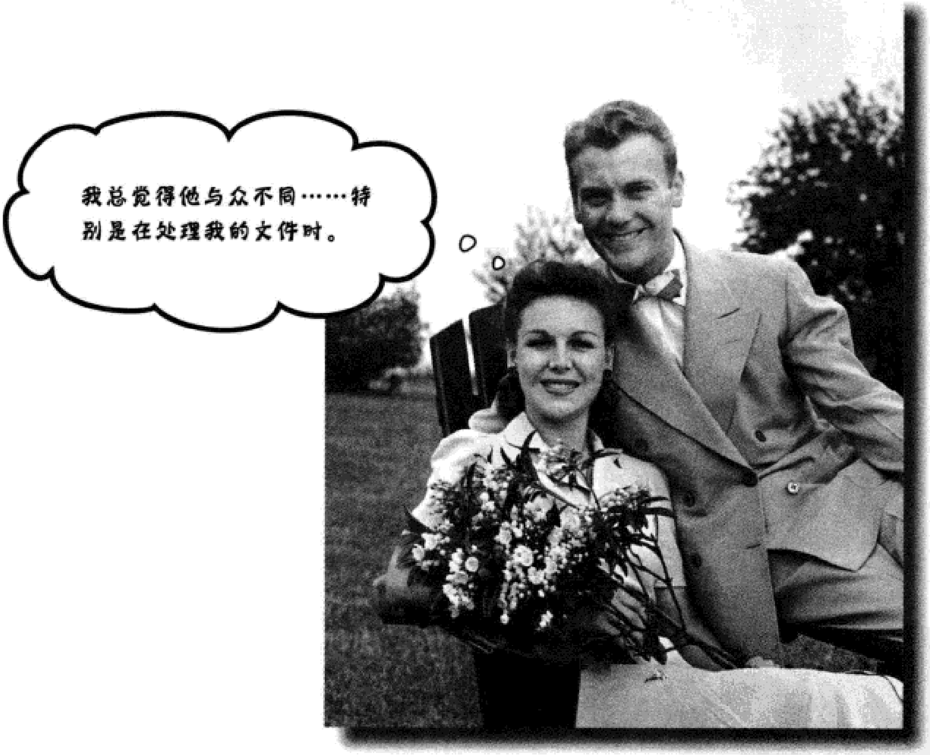
- 在IDLE编辑窗口中按F5可以“运行”代码。
- 按F5将模块的代码“加载”到IDLE shell，模块名会专门导入到IDLE的命名空间。使用IDLE时这很便利。在代码中，需要显式使用import语句。

BULLET POINTS

- 模块是一个包含Python代码的文本文件。
- 发布工具允许将模块转换为可共享的包。
- `setup.py`程序提供了模块的元数据，用来构建、安装和上传打包的发布。
- 使用import语句可以将模块导入到其他程序中。
- Python中的各个模块提供了自己的命名空间，使用`module.function()`形式调用模块的函数时，要用命名空间名限定函数。
- 使用import语句的`from module import function`形式可以从一个模块将函数专门导入到当前命名空间。
- 使用#可以注释掉一行代码，或者为程序增加一个简短的单行注释。
- 内置函数（built-in functions，BIF）有自己的命名空间，名为`__builtins__`，这会自动包含在每一个Python程序中。
- `range()` BIF可以与for结合使用，从而迭代固定次数。
- 包含`end=''`作为`print()` BIF的一个参数会关闭其默认行为（即在输入中自动包含换行）。
- 如果为函数参数提供一个缺省值，这个函数参数就是可选的。

3 文件与异常

处理错误

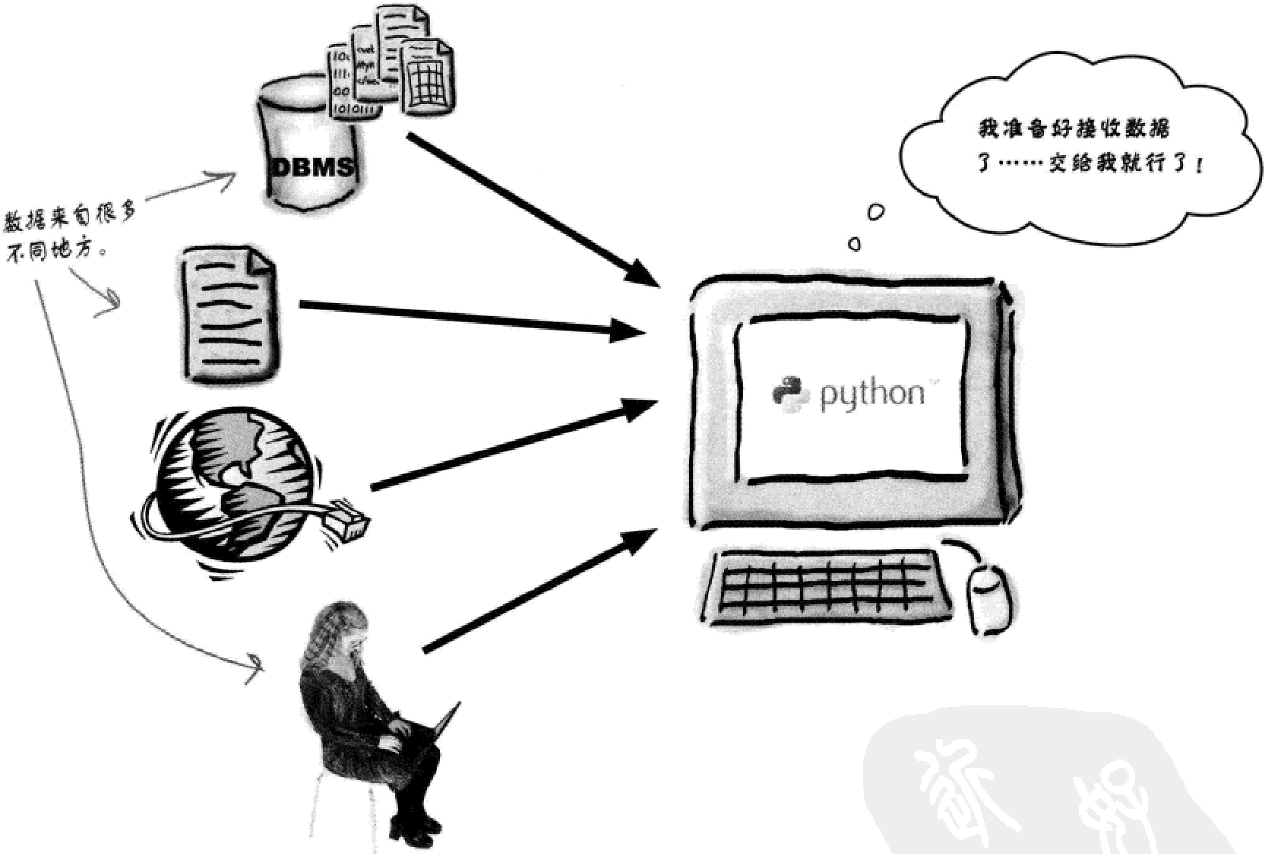


只是在代码中处理列表数据还不够。

你还需要把数据轻松地送入程序。由此说来，不难理解为什么利用Python可以很容易地从文件读取数据。这一点很棒，不过再想想看，与程序之外的数据交互时可能有麻烦……另外还有方方面面的问题在等着给你下绊！出问题时，需要一种策略使你能够从麻烦中脱身，利用Python的异常处理机制来处理异常情况就是这样一种策略，这一章将会介绍这种机制。

程序外部的数据

大多数程序都遵循输入-处理-输出模型：首先输入数据，进行处理，然后存储、显示、打印或传输。



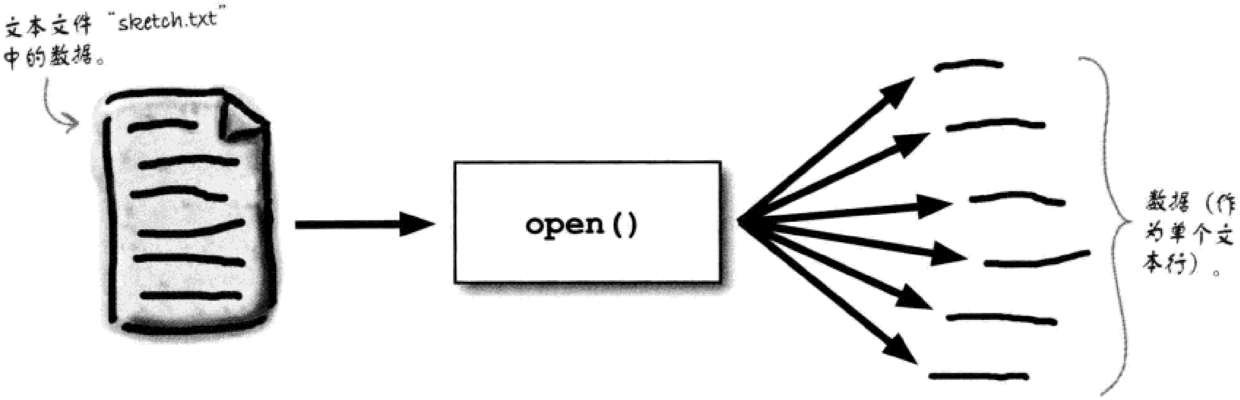
到目前为止，你已经了解如何处理数据，也知道如何在屏幕上显示数据。不过你知道如何向程序传入数据吗？具体来讲，怎样从文件读取数据？

Python如何从文件读取数据？

都是文本行

Python中的基本输入机制是基于行的：从文本文件向程序读入数据时，一次会到达一个数据行。

Python的`open()` BIF就是用来与文件交互。如果与for语句结合使用，可以非常容易地读取文件。



使用`open()` BIF处理文件中的数据时，会创建一个迭代器从文件向你的代码输入数据行，一次传入一行数据。不过先别着急。对现在来说，先来考虑Python中的标准"打开-处理-关闭"代码：

打开……

……处理……

……关闭。

```
the_file = open('sketch.txt')
# Do something with the data
# in "the_file".
the_file.close()
```

动手做！

创建一个文件夹HeadFirstPython，另外创建一个子文件夹chapter3。有了这些文件夹后，从Head First Python支持网站下载sketch.txt，把它保存到chapter3文件夹。

使用IDLE来感受Python的文件输入机制。

An IDLE Session

打开一个新的IDLE会话，导入os模块，并把当前工作目录切换到包含刚下载的数据文件的那个文件夹：

```
>>> import os                    ← 从标准库导入"os"。
>>> os.getcwd()                  ← 当前工作目录是什么？
'/Users/barryp/Documents'
>>> os.chdir('../HeadFirstPython/chapter3')   ← 切换为包含数据文件的文件夹。
>>> os.getcwd()                  ← 确认现在在正确的目录下。
'/Users/barryp/HeadFirstPython/chapter3'
```

现在，打开数据文件，从文件读取前两行，并在屏幕上显示出来：

```
>>> data = open('sketch.txt')    ← 打开一个命名文件，将文件赋至一个名为"data"的文件对象。
>>> print(data.readline(), end='')
Man: Is this the right room for an argument?
>>> print(data.readline(), end='')
Other Man: I've told you once.
```

使用"readline()"方法从文件获取一个数据行，然后使用"print()"BIF在屏幕上显示这个数据行。

下面再"退回"到文件起始位置，然后使用for语句处理文件中的每一行：

```
>>> data.seek(0)     ← 使用"seek()"方法返回到文件起始位置。当然，对Python的文件也可以使用"tell()"。
0
>>> for each_line in data:
        print(each_line, end='')
```

这个代码看上去应该很熟悉：这是一个标准的迭代，这里使用了文件数据作为输入。

```
Man: Is this the right room for an argument?
Other Man: I've told you once.
Man: No you haven't!
Other Man: Yes I have.
Man: When?
Other Man: Just now.
Man: No you didn't!
  ......
Man: (exasperated) Oh, this is futile!!
(pause)
Other Man: No it isn't!
Man: Yes it is!
>>> data.close()     ← 既然已经处理完文件，一定要将它关闭。
```

每个数据行都显示在屏幕上（不过由于篇幅原因，这里做了删减）。

进一步查看数据

再来进一步查看数据。看起来它遵循某种特定的格式：

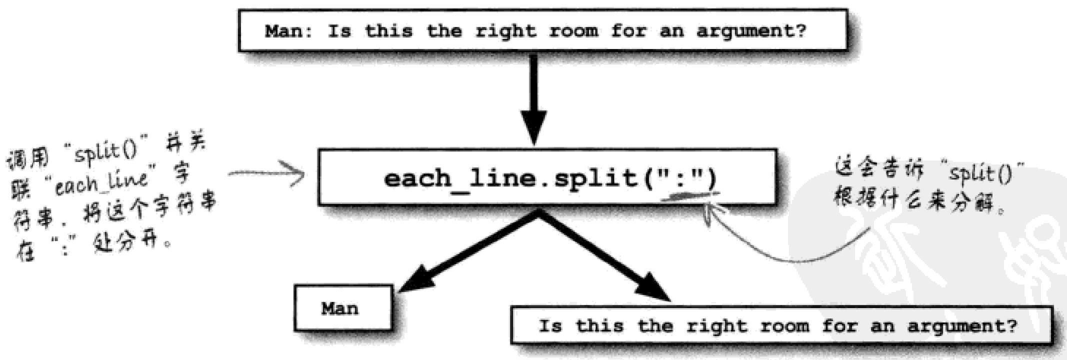
演员角色：

后面是一个冒号，然后是一个空格符。

演员讲的台词。

```
Man: Is this the right room for an argument?
Other Man: I've told you once.
Man: No you haven't!
Other Man: Yes I have.
Man: When?
Other Man: Just now.
Man: No you didn't!
```

了解这个格式后，可以处理各行数据，根据需要抽取出数据行中的各个部分。在这里`split()`方法可以提供帮助：



split()方法返回一个字符串列表，这会赋至一个目标标识符列表。这称为多重赋值（multiple assignment）：

左边是一个目标标识符列表……

……赋为"split()"返回的各个字符串。

```
(role, line_spoken) = each_line.split(":")
```

对于上面的示例数据，"role"会赋值为字符串"Man"，而……

……"line_spoken"赋值为字符串"Is this the right room for an argument?"

嗯，真的吗？☺

An IDLE Session

下面来确认还能处理文件同时分解各行，在IDLE shell中键入以下代码：

```
>>> data = open('sketch.txt')
>>> for each_line in data:
        (role, line_spoken) = each_line.split(':')
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')

Man said:  Is this the right room for an argument?
Other Man said:  I've told you once.
Man said:  No you haven't!
Other Man said:  Yes I have.
Man said:  When?
Other Man said:  Just now.
Man said:  No you didn't!
Other Man said:  Yes I did!
Man said:  You didn't!
Other Man said:  I'm telling you, I did!
Man said:  You did not!
Other Man said:  Oh I'm sorry, is this a five minute argument, or the full half hour?
Man said:  Ah! (taking out his wallet and paying) Just the five minutes.
Other Man said:  Just the five minutes. Thank you.
Other Man said:  Anyway, I did.
Man said:  You most certainly did not!
Traceback (most recent call last):
  File "<pyshell#10>", line 2, in <module>
    (role, line_spoken) = each_line.split(':')
ValueError: too many values to unpack
```

打开数据文件。

处理数据，从每个数据行抽取各个部分，并在屏幕上显示。

这些看起来都很正常。

唉呀！这里出了严重的错误。



这是一个ValueError错误，说明文件中的数据肯定有问题，是吗？

了解你的数据

你的代码刚开始还能正常工作，然后就崩溃了，指示有一个运行时错误。问题就出在Man说“You most certainly did not!”那行数据后面。

下面来看数据文件，看看这行得到成功处理的数据后面是什么：

```
Man: You didn't!
Other Man: I'm telling you, I did!
Man: You did not!
Other Man: Oh I'm sorry, is this a five minute argument, or the full half hour?
Man: Ah! (taking out his wallet and paying) Just the five minutes.
Other Man: Just the five minutes. Thank you.
Other Man: Anyway, I did.
Man: You most certainly did not!
Other Man: Now let's get one thing quite clear: I most definitely told you!
Man: Oh no you didn't!
Other Man: Oh yes I did!
```

错误就出在这行数据后面。

注意到什么没有？

注意到下一行数据有什么问题吗？

下一行数据有两个冒号，而不是一个。正是这个多余的数据导致split()方法无法正常工作，因为按你目前的代码来看，split()要把这个数据行分为两部分，分别赋给role和line _ spoken。

数据中出现一个额外的冒号时，split()方法将这一行分为3个部分。你的代码没有告诉split()该如何处理第3部分，所以Python解释器会产生一个ValueError，抱怨“值过多”，然后终止。这就出现了一个运行时错误。

BRAIN POWER

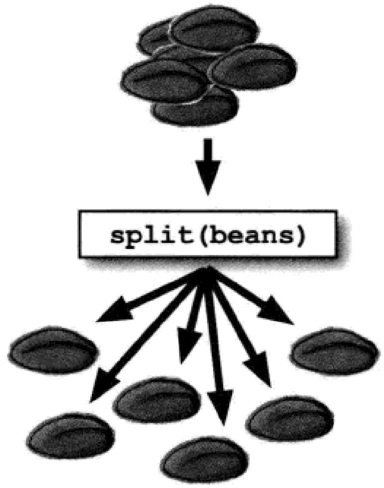
你会采用什么方法来解决这个数据处理问题？

动手做！

为了帮助诊断这个问题，下面把代码放在一个单独的文件中，名为sketch.py。可以从IDLE shell把代码复制粘贴到一个新的IDLE编辑窗口中。

了解你的方法，请求帮助

可以看看split()方法是否包含对你有帮助的功能，这可能很有用。可以使用help() BIF让IDLE shell告诉你更多有关split()方法的信息。



An IDLE Session

```
>>> help(each_line.split)
Help on built-in function split:

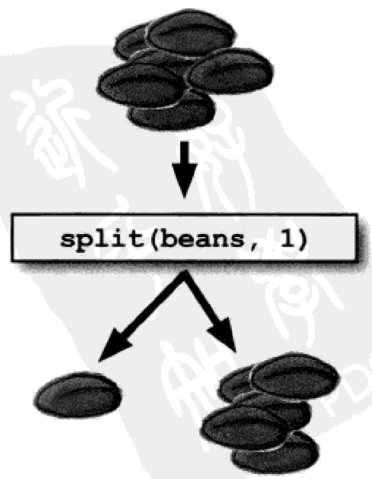
split(……)
    S.split([sep[, maxsplit]]) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
    removed from the result.
```

看起来"split()"有一个可选的参数。

split()的这个可选参数控制着将数据行分解为多少个部分。默认地，数据会尽可能多地分解。不过你只需要两个部分：角色名和他讲的台词。

如果将这个可选参数设置为1，数据行只会分解为两部分，这实际上就消除了数据行中额外的冒号的影响。



下面来试一试，看看会发生什么。

Geek Bits

IDLE 提供了Help→Python Docs菜单选项（这会在你的Web浏览器中打开文档），可以利用这个菜单搜索整个Python文档。如果你只想看某个方法或函数有关的文档，可以在IDLE shell中使用help() BIF。

An IDLE Session

以下是IDLE编辑窗口中的代码。注意split()方法的额外参数。

sketch.py - /Users/barryp/HeadFirstPython/chapter4/sketch.py

```
data = open('sketch.txt')

for each_line in data:
    (role, line_spoken) = each_line.split(':', 1)
    print(role, end='')
    print(' said: ', end='')
    print(line_spoken, end='')

data.close()
```

Ln: 11 Col: 0

这个额外的参数控制"split()"如何分解。

完成编辑并保存后，按F5（或者从IDLE的Run菜单选择Run Module），尝试运行这个版本的代码：

```
>>> ================================ RESTART ================================
>>>
Man said:  Is this the right room for an argument?
Other Man said:  I've told you once.
Man said:  No you haven't!
Other Man said:  Yes I have.
Man said:  When?
Other Man said:  Just now.
    ...
Other Man said:  Anyway, I did.
Man said:  You most certainly did not!
Other Man said:  Now let's get one thing quite clear: I most definitely told you!
Man said:  Oh no you didn't!
Other Man said:  Oh yes I did!
Man said:  Oh no you didn't!
Other Man said:  Oh yes I did!
Man said:  Oh look, this isn't an argument!
Traceback (most recent call last):
  File "/Users/barryp/HeadFirstPython/chapter4/sketch.py", line 5, in <module>
    (role, line_spoken) = each_line.split(':', 1)
ValueError: need more than 1 value to unpack
```

这里显示的输出有所删减，以保证这一页上能放下重要的内容。

太棒了。包含两个冒号的数据行可以顺利通过了……

……不过快乐很短暂。又有一个ValueError!!

这足以毁掉你一天的好心情。现在又哪里出错了呢？

更好地了解你的数据

你的代码又产生一个`ValueError`，不过这一次并不是抱怨“值过多”，Python解释器在抱怨没有足够的数据来处理：“need more than 1 value to unpack”（需要不只一个值）。希望再通过一次快速查看可以澄清缺少数据的秘密。

```
Other Man: Now let's get one thing quite clear: I most definitely told you!
Man: Oh no you didn't!
Other Man: Oh yes I did!
Man: Oh no you didn't!
Other Man: Oh yes I did!
Man: Oh look, this isn't an argument!
(pause)
Other Man: Yes it is!
Man: No it isn't!
(pause)
Man: It's just contradiction!
Other Man: No it isn't!
```

这是什么？有些数据不符合我们期望的格式……这可不好。

缺少冒号的情况

有些数据行不包含冒号，`split()`方法查找冒号时就会出现问题。由于缺少冒号，`split()`无法完成它的工作，因此导致了运行时错误，因此会抱怨解释器需要“不只一个值”。



两种截然不同的方法



Jill建议的方法肯定是可行的：增加必要的额外逻辑来确定是否可以在数据行上调用`split()`。所要做的就是明确如何检查数据行。

Joe的方法也是可行的：让错误出现，监视它的发生，然后从运行时错误（以某种方式）恢复。

在这里哪一种方法更好呢？

增加额外逻辑

下面每种方法都来尝试一下，然后再决定这里哪一种方法更适用。

除了split()方法，每个Python字符串还有一个find()方法。可以让find()来尝试找出一个字符串中的子串，如果无法找到，find()方法会返回值-1。如果find()可以找到子串，这个方法会返回该子串在原字符串中的索引位置。

An IDLE Session

将一个字符串赋至each_line变量，其中不包含冒号，然后使用find()尝试查找冒号：

```
>>> each_line = "I tell you, there's no such thing as a flying circus."
>>> each_line.find(':')
-1
```

这个字符串不包含冒号，所以"find()"返回-1表示未找到。

按两次Alt-P，回退到为变量赋字符串的那行代码，不过这一次要编辑这个字符串，让它包含一个冒号，然后再使用find()方法尝试查找冒号：

```
>>> each_line = "I tell you: there's no such thing as a flying circus."
>>> each_line.find(':')
10
```

这个字符串确实包含一个冒号，所以"find()"返回一个正的索引值。



Exercise

调整你的代码，使用上一页展示的额外逻辑技术来处理不包含冒号字符的数据行。

这里需要加什么条件？

```
data = open('sketch.txt')

for each_line in data:
    if ...............................................
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')

data.close()
```

Sharpen your pencil

你能想到这种技术可能存在的其他问题吗？拿出笔来，在下面给出的空白处写出这种方法可能遇到的问题：

Exercise Solution

调整你的代码，使用额外逻辑技术来处理不包含冒号字符的数据行。

```
data = open('sketch.txt')

for each_line in data:
    if not each_line.find(':') == -1:
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')

data.close()
```

理解这个条件需要花点时间，不过这确实能奏效。

注意这里使用了"not"关键字，这会将条件的值取反。

Sharpen your pencil Solution

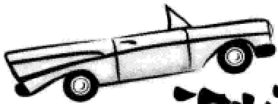
你能想到这种技术可能存在的其他问题吗？拿出笔来，在下面给出的空白处写出这种方法可能遇到的问题：

如果你写出的问题与这里不完全相同也没有关系，只要与此类似就可以了。

如果数据文件的格式发生改变，这个代码可能会有问题，相应地也需要改变条件。

if语句使用的条件有点不好读，也不好理解。

这个代码有点"脆弱"……如果再出现另外一个异常情况它就会出问题。



测试驱动

在IDLE的编辑窗口中修改代码，按下F5看它能不能正常工作。

```
Python Shell
>>> ================================ RESTART ================================
>>>
Man said:  Is this the right room for an argument?
Other Man said:  I've told you once.
Man said:  No you haven't!
Other Man said:  Yes I have.
Man said:  When?
Other Man said:  Just now.
Man said:  No you didn't!
Other Man said:  Yes I did!
Man said:  You didn't!
Other Man said:  I'm telling you, I did!
Man said:  You did not!
Other Man said:  Oh I'm sorry, is this a five minute argument, or the full half hour?
Man said:  Ah! (taking out his wallet and paying) Just the five minutes.
Other Man said:  Just the five minutes. Thank you.
Other Man said:  Anyway, I did.
Man said:  You most certainly did not!
Other Man said:  Now let's get one thing quite clear: I most definitely told you!
Man said:  Oh no you didn't!
Other Man said:  Oh yes I did!
Man said:  Oh no you didn't!
Other Man said:  Oh yes I did!
Man said:  Oh look, this isn't an argument!
Other Man said:  Yes it is!
Man said:  No it isn't!
Man said:  It's just contradiction!
Other Man said:  No it isn't!
Man said:  It IS!
Other Man said:  It is NOT!
Man said:  You just contradicted me!
Other Man said:  No I didn't!
Man said:  You DID!
Other Man said:  No no no!
Man said:  You did just then!
Other Man said:  Nonsense!
Man said:  (exasperated) Oh, this is futile!!
Other Man said:  No it isn't!
Man said:  Yes it is!
>>>
Ln: 149 Col: 4
```

这一次没错误了。

你的程序可以正常工作了……尽管有点脆弱。

如果文件的格式改变，你的代码也需要改变，而且更多代码通常意味着更大的复杂性。增加额外代码来处理异常情况的做法确实可行，但是从长远来看这是有代价的。



处理异常

注意到了吗？代码出问题时，Python解释器会显示一个traceback，后面跟着一个错误消息。

Python通过traceback来告诉你运行时发生了某种意外情况。在Python世界里，运行时错误叫做异常（exception）。



当然，如果你决定在异常出现时将其忽略，你的程序将会崩溃，

不过事实上，Python允许你在异常发生时捕获异常，这样就为你提供了一个机会，可以从这个错误中恢复，最重要的是可以避免崩溃。

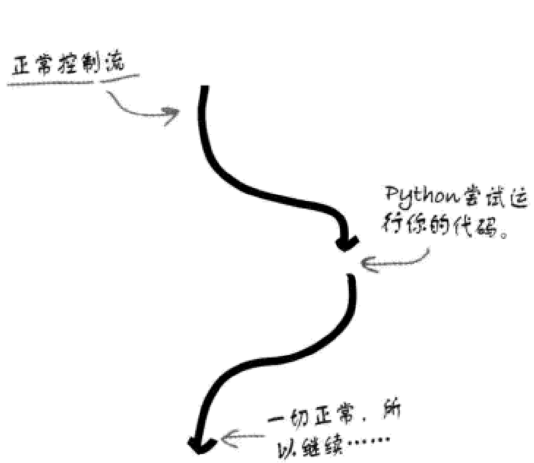
通过控制程序的运行时行为，你可以（尽可能地）确保你的Python程序在面对大多数运行时错误时是健壮的。

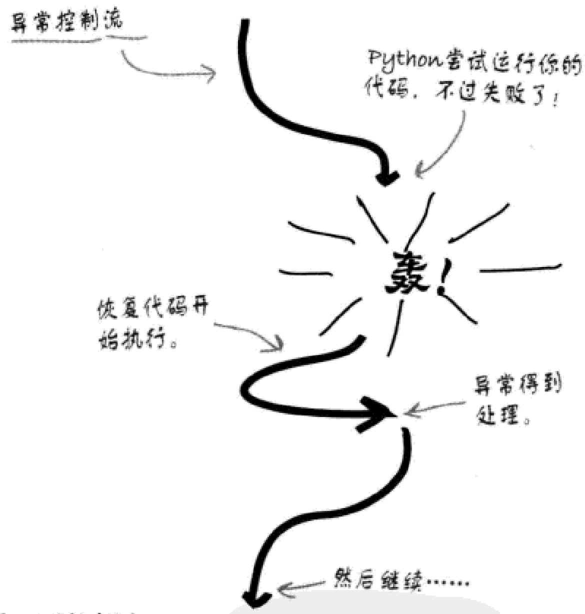
先尝试运行代码，然后处理可能发生的错误。

先尝试，然后恢复

并非增加额外的代码和逻辑来阻止不好的事情发生，Python的异常处理机制允许错误出现，但监视它的发生，然后给你一个机会来恢复。

在正常的控制流期间，Python尝试运行你的代码，如果没有任何问题，代码会继续正常执行。在异常控制流期间，Python先尝试运行你的代码，如果发现有问题，就会执行恢复代码，然后继续正常执行你的代码。



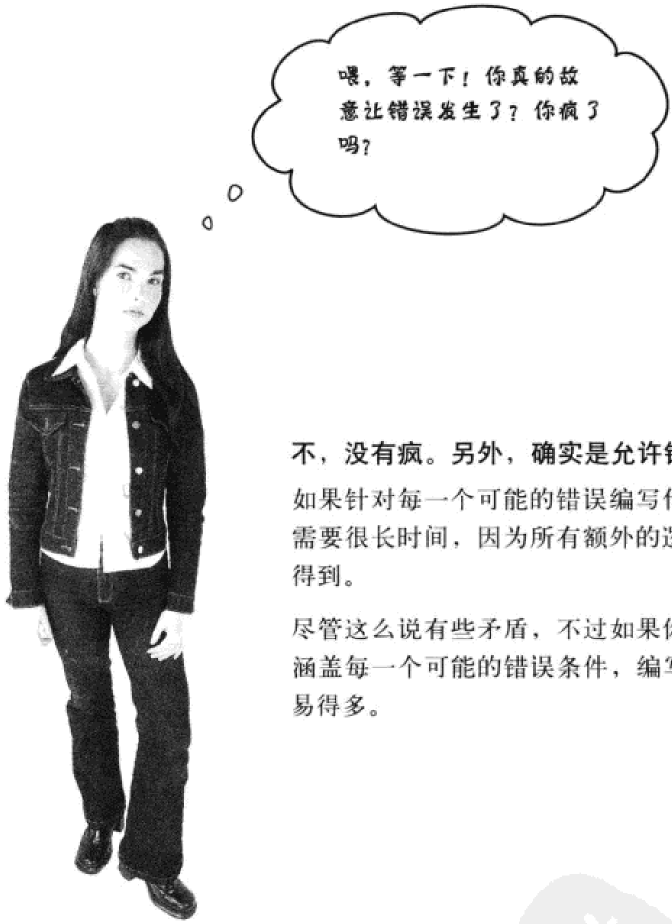


try/except机制

Python有一个try语句，这个语句就是为了向你提供一个途径，可以在运行时系统地处理异常和错误。try语句的一般形式如下：

```
try:
    你的代码（可能导致一个运行时错误）
except:
    错误恢复代码
```

"try"和"except"都是Python关键字。



不，没有疯。另外，确实是允许错误发生。

如果针对每一个可能的错误编写代码，编写这个代码可能需要很长时间，因为所有额外的逻辑需要花一番工夫才能得到。

尽管这么说有些矛盾，不过如果你操心得越少，不是希望涵盖每一个可能的错误条件，编写代码的任务实际上会容易得多。

找出要保护的代码

为了深入了解Python异常处理机制，先花点时间找出需要保护的代码。

Sharpen your pencil

研究你的程序，圈出你认为需要保护的代码行（可能是多行代码）。然后，在下面给出的空白处说明为什么要保护这些代码。

```
data = open('sketch.txt')

for each_line in data:
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')

data.close()
```

在这里写出你的原因。

there are no Dumb Questions

问： 有个问题困扰我很久了。split()方法执行时，它传回一个列表，不过目标标识符包围在小括号之间，而不是中括号，这真的是一个列表吗？

答： 你观察得很仔细。Python中实际上有两种类型的列表：一种是可以改变的列表（用中括号包围），另一种一旦创建就不能改变（用小括号包围）。后者是一种不可变列表，更常见的称呼是元组（tuple）。可以认为元组等同于列表，不过有一点区别：一旦创建，元组中的数据在任何情况下都不能改变。还有一种理解，可以认为元组是一个"常量列表"，在Head First，我们把"tuple"拼作与"couple"押韵，也有人把"tuple"拼作与"rupal"押韵。至于哪一种正确，并没有一个明确的共识，所以你可以选择并采用任意一种方式。

Sharpen your pencil Solution

研究你的程序，圈出你认为需要保护的代码行（可能是多行代码）。然后，在下面给出的空白处说明为什么要保护这些代码。

```
data = open('sketch.txt')

for each_line in data:
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')

data.close()
```

这4行代码都需要保护。

如果"split()"调用失败，你不希望下面的3个"print()"语句执行，所以最好保护"if"组中的所有4行代码，而不只是调用"split()"的那一行代码。



放过错误

对于这个数据（和程序），最好能忽略不符合期望格式的数据行。如果split()方法调用导致一个异常，可以报告这是一个错误并使用**pass**继续执行代码。

如果遇到这样一种情况，也就是希望你提供一些代码，但是并不需要具体做什么，就可以使用Python的pass语句（可以把它认为是空语句或 null语句）。

下面是结合try使用的pass语句：

```
data = open('sketch.txt')

for each_line in data:
    try:
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')
    except:
        pass

data.close()
```

这个代码得到保护，可以避免运行时错误。

如果出现一个运行时错误，会执行这个代码。

现在，不论调用split()方法时发生了什么，try语句会捕获所有异常并处理，用pass忽略这个错误。

下面来看这个代码的实际运行。

动手做！

在IDLE编辑窗口中对代码做必要的修改。

An IDLE Session

在IDLE编辑窗口中修改代码，按下F5来运行。

sketch-try.py - /Users/barryp/HeadFirstPython/chapter4/sketch-try.py

```
data = open('sketch.txt')

for each_line in data:
    try:
        (role, line_spoken) = each_line.split(':', 1)
        print(role, end='')
        print(' said: ', end='')
        print(line_spoken, end='')
    except:
        pass

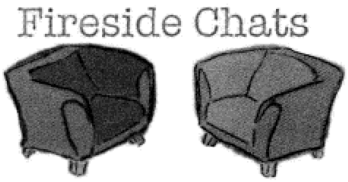
data.close()
```

Ln: 14 Col: 0

```
>>> ================================ RESTART ================================
>>>
Man said:  Is this the right room for an argument?
Other Man said:  I've told you once.
Man said:  No you haven't!
Other Man said:  Yes I have.
Man said:  When?
    ...
OOther Man said:  Nonsense!
Man said:  (exasperated) Oh, this is futile!!
Other Man said:  No it isn't!
Man said:  Yes it is!
```

这个代码可以正常工作，而且没有运行时错误。





今晚话题：用额外代码还是异常处理程序处理运行时错误

额外代码：

通过确保运行时错误根本不会发生，我能保证代码充分安全，绝对不会遭遇traceback。

复杂性又不会伤害任何人。

我真是搞不懂。看到你的代码在你面前“爆炸”，你居然还更高兴……你是不是觉得等火着起来再灭火很不错？

但是不好的事情总是会发生在你头上。我就不会遇到这种事情，因为我根本不允许它们发生。

嗯……那要看情况。如果你足够聪明（相信我，我就很聪明），完全可以考虑到所有可能的运行时问题，并写好代码避免这些问题。

努力工作总不是坏事。

我的代码当然都是需要的！否则怎么避开所有那些可能发生的运行时错误呢？

嗯，嗯……我想起码是大部分错误。

恶狠狠地瞪着：拜托，住口！

异常处理程序：

但是要以增加复杂性为代价……

我想我得提醒你，下一次不要再在凌晨4点的时候还在调试一段复杂的代码。

没错。我关注的是首先完成我的工作。如果有不好的情况发生，我会做好准备。

除非发生你意想不到的事情。那样一来你就完蛋了。

在我听来好像要做很多很多额外的工作吧。

你听到我之前说过凌晨4点的时候还在调试那件事，是吧？有时我想你实际上就是喜欢写一些你并不需要的代码……

那么……到底有多少个呢？

你并不知道，不是吗？你根本不知道倘若出现一个未知或意料外的运行时错误会发生什么，对不对？

其他错误呢？

这两种方法确实都可行，这一点不假，不过再来考虑遇到其他错误时会发生什么。



处理缺少的文件

Frank提出一个很有意思的问题，当然，由于数据文件被删除而导致的问题会让Jill和Joe日子更不好过。缺少数据文件时，这两个版本的程序都会崩溃，产生一个IOError。

动手做！ →

将数据文件重命名，然后再运行这两个版本的程序，确认它们确实会产生一个IOError并生成一个traceback。

增加更多错误检查代码……

如果你是“不要让错误发生”那一派的忠实追随者，你的第一个反应肯定是再增加另外的代码，在尝试打开文件之前先查看数据文件是否存在，是不是？

下面就来实现这个策略。Python的os模块有一些工具可以帮助确定一个数据文件是否存在，所以我们需要从标准库导入这个模块，然后向代码增加必要的检查：

```
import os

if os.path.exists('sketch.txt'):
    data = open('sketch.txt')

    for each_line in data:
        if not each_line.find(':') == -1:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said: ', end='')
            print(line_spoken, end='')

    data.close()
else:
    print('The data file is missing!')
```

检查文件是否存在。

所有这些代码都保持不变。

通知用户坏消息。

An IDLE Session

简单测试这个代码来确认这个新问题已经得到妥善解决。在IDLE的编辑窗口中完成这个新版本的代码后，按下F5确认一切正常。

```
>>> ================================ RESTART ================================
>>>
The data file is missing!
>>>
```

正如所料。太棒了。

……或者再增加一层异常处理

如果你是“异常发生时再做处理”那一派的忠实追随者，你会把代码包在另一个try语句中。

增加另一个“try”语句。

与前一个程序一样，所有这些代码都保持不变。

告诉用户坏消息。

```
try:
    data = open('sketch.txt')

    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said: ', end='')
            print(line_spoken, end='')
        except:
            pass

    data.close()
except:
    print('The data file is missing!')
```

An IDLE Session

需要再做一个简单的测试，这一次测试这个使用异常处理的程序版本。按下F5来运行。

```
>>> ================================ RESTART ================================
>>>
The data file is missing!
>>>
```

不出所料，这个版本的程序也能处理缺少文件的情况。

那么，哪种方法更好呢？

嗯……这要看你问的是谁！下面给出这两个版本的代码：

sketch-extra-logic-file-missing.py - /Users/barryp/HeadFirstPython/chapter3/sk...

```
import os

if os.path.exists('sketch.txt'):
    data = open('sketch.txt')

    for each_line in data:
        if not each_line.find(':') == -1:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said ', end='')
            print(line_spoken, end='')

    data.close()
else:
    print('The datafile is missing!')
```

这个版本使用额外的逻辑来处理文件I/O错误。

sketch-with-try-file-missing.py - /Users/barryp/HeadFirstPython/chapter3/sket...

```
try:
    data = open('sketch.txt')

    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said ', end='')
            print(line_spoken, end='')
        except:
            pass

    data.close()
except:
    print('The datafile is missing!')
```

Ln: 17 Col: 0

这个版本使用另一个"try"语句来处理文件I/O错误。

下面对这两个版本的程序问一个简单的问题：这两个程序分别做了什么？

Sharpen your pencil

拿出笔来。在第一个方框中写出你认为左边的程序做了什么。在第2个方框中写出你认为右边的程序做了些什么。

1 ....

....

....

2 ....

....

....

Sharpen your pencil Solution

拿出笔来。在第一个方框中写出你认为左边的程序做了什么。在第2个方框中写出你认为右边的程序做了些什么。

要写的有很多，所以上一页留出的空间并不够，你还需要更大空间。

1. 左边的代码首先导入“os”库，然后使用“path.exists”来确保数据文件存在，在此之后才尝试打开这个数据文件。然后处理文件中的每一行，不过只是在确定数据行符合所需的格式之后才进行处理，这是通过检查数据行中是否有一个“:”字符来做到的。如果找到了“:”，则处理这个数据行；否则就将这个数据行忽略。所有工作完成时，关闭数据文件。如果文件未找到，最后你会得到一个友好的消息。

现在……这里留的空间大小还算合适。

2. 右边的代码打开一个数据文件，处理这个文件中的各个数据行，抽取出感兴趣的数据，并显示在屏幕上。完成后关闭文件。如果出现任何异常，这个代码会进行处理。

复杂性通常不是个好东西

看到这里发生什么了吗？

随着你必须考虑的错误越来越多，“增加额外代码和逻辑”方案的复杂性也随之增加，直到最后可能会掩盖程序的本来作用。

而异常处理方案就不存在这个问题，可以一目了然地看出程序的主要作用。

通过使用Python的异常处理机制，你可以关注代码真正需要做什么，而不必操心哪里可能出问题，并编写额外的代码来避免运行时错误。

谨慎地使用`try`语句可以让代码更易读，更易写，而且可能这是最重要的——出问题时更容易修正。

要重点关注你的代码需要做什么。

大功告成……不过还有一个小问题

你的异常处理代码很不错。实际上，你的代码非常好，因为它很有一般性。

现在，不论运行时出现什么错误，都会由代码处理，因为这个错误将被忽略，或者显示一个错误消息。不过，你真正需要考虑的只是IOError和ValueError，因为这些才是之前你开发程序时出现的异常类型。

能处理所有运行时错误固然很好，但是过于一般化可能有些不明智……你可能想知道代码在运行时执行时是否还会出现IOError或ValueError以外的其他异常。如果确实发生了其他问题，你的代码可能会以一种不合适的方式处理。

```
try:
    data = open('sketch.txt')

    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said: ', end='')
            print(line_spoken, end='')
        except:
            pass

    data.close()
except:
    print('The data file is missing!')
```

尝试运行的代码中出现任何运行时错误时都会运行这个代码和下面这个代码。

现在编写的代码过于一般化。出现的任何运行时错误都会由某个except组处理。这可能并不是你希望的，因为这个代码会悄无声息地忽略运行时错误。

需要以一种不那么一般化的方式使用except。

特定指定异常

如果你的异常处理代码设计为处理一种特定类型的错误，一定要在except代码行上指定错误类型。这样一来，就可以把一般化的异常处理代码转变为具有特定性。

```
try:
    data = open('sketch.txt')

    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            print(role, end='')
            print(' said: ', end='')
            print(line_spoken, end='')
        except ValueError:
            pass

    data.close()
except IOError:
    print('The data file is missing!')
```

指定要处理的运行时错误类型。

当然，如果出现一个不同类型的运行时错误，你的代码将不再处理这个错误，但是至少现在你对这个异常已经有所认识。如果特别指定了代码要处理某种特定的运行时错误，你的程序就不会再悄无声息地忽略某些运行时错误了。





你的Python工具箱

你已经读完了第3章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- “异常”（exception）因运行时错误而出现，会产生一个traceback。
- “traceback”是出现的运行时错误的一个详细描述。

IDLE说明

- 可以从IDLE的Help菜单选择Python Docs来访问Python的文档。这会在你喜欢的Web浏览器中打开Python 3文档。

BULLET POINTS

- 使用`open()` BIF打开一个磁盘文件，创建一个迭代器从文件读取数据，一次读取一个数据行。
- `readline()`方法从一个打开的文件读取一行数据。
- `seek()`方法可以用来将文件“退回”到起始位置。
- `close()`方法关闭一个之前打开的文件。
- `split()`方法可以将一个字符串分解为一个子串列表。
- Python中不可改变的常量列表称为元组（tuple）。一旦将列表数据赋至一个元组，就不能再改变。元组是不可改变的。
- 数据不符合期望的格式时会出现`ValueError`。
- 数据无法正常访问时会出现`IOError`（例如，可能你的数据文件已经被移走或者重命名）。
- `help()` BIF允许你在IDLE shell中访问Python的文档。
- `find()`方法会在一个字符串中查找一个特定子串。
- `not`关键字将一个条件取反。
- `try/except`语句提供了一个异常处理机制，从而保护可能导致运行时错误的某些代码行。
- `pass`语句就是Python的空语句或null语句，它什么也不做。

4 持久存储

数据保存到文件

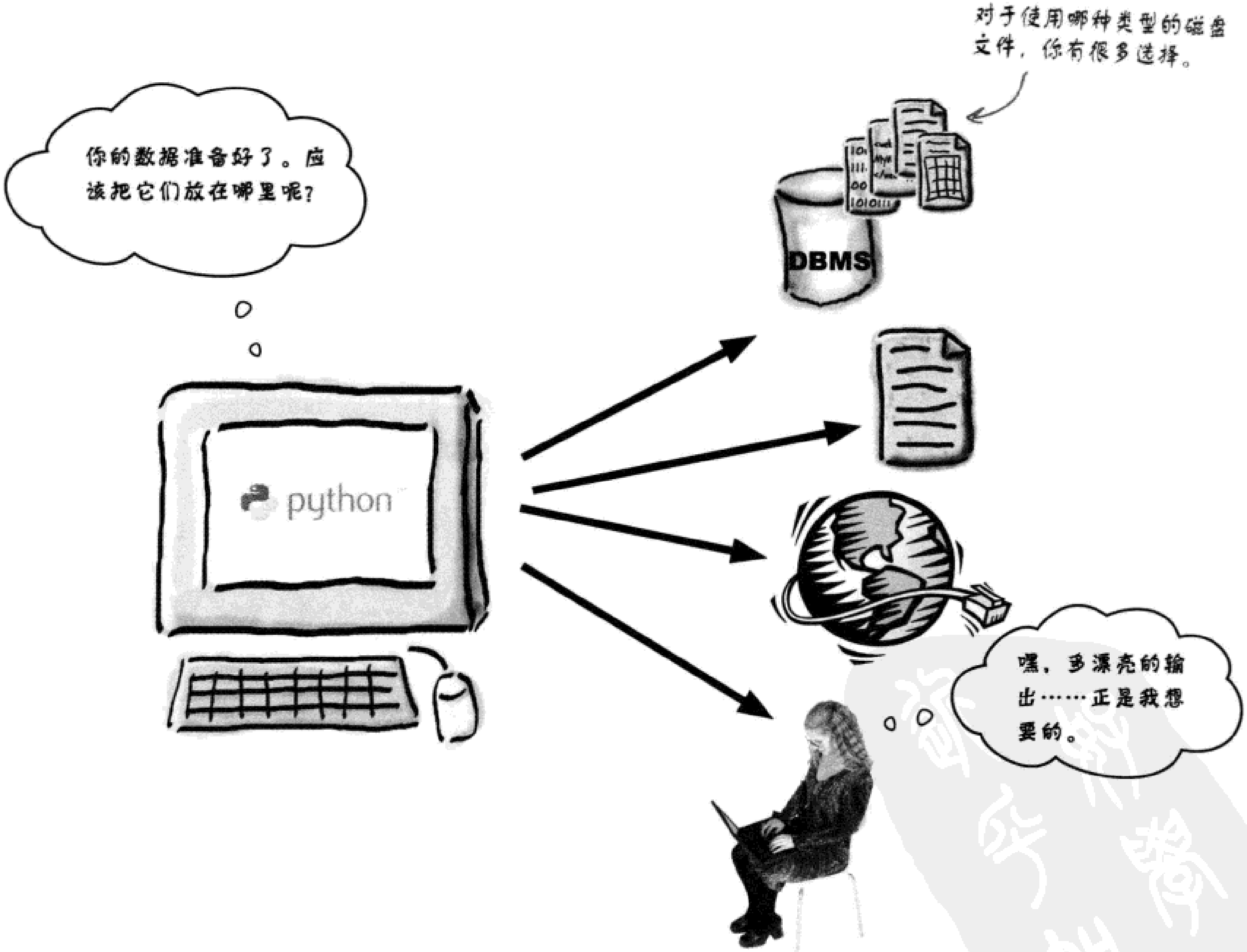


能够处理基于文件的数据确实很棒。

但是工作完成时你的数据会怎么样呢？当然，最好把数据保存到一个磁盘文件中，这样就能在以后某个时间再次使用这些数据。将基于内存的数据存储到磁盘上，这正是持久存储的含义。Python支持所有将数据写至文件的常用工具，另外还提供了一些很酷的工具,可以高效地存储Python数据。所以……请翻开下一页，下面开始学习这些内容。

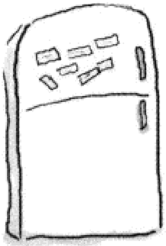
程序生成数据

很少会有程序从一个磁盘文件读取并处理数据后就把处理后的数据丢掉。通常情况下，程序会保存所处理的数据，将输出显示在屏幕上，或者通过网络传输数据。



学习如何将数据写入磁盘之前，先来处理上一章的数据，看看谁对谁说了什么。

完成这个工作之后，你会得到一些值得保存的结果。



代码磁贴

把这一页最下面的代码磁贴加到现有的代码中，需要满足以下要求：

1. 创建一个空列表，名为man。
2. 再创建一个空列表，名为other。
3. 增加一行代码，删除line_spoken变量中不需要的空白符。
4. 给出条件和代码，根据role的值将line_spoken增加到适当的列表中。
5. 在屏幕上输出各个列表（man和other）。

```
..........................

..........................
try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)

            ..........................................................

            ............................

                ............................................

            ............................

                ............................................

        except ValueError:
            pass
    data.close()
except IOError:
    print('The datafile is missing!')

..........................

..........................
```

这里是你要用的磁贴。

```
elif role == 'Other Man':
```

```
man = []
```

```
if role == 'Man':
```

```
other.append(line_spoken)
```

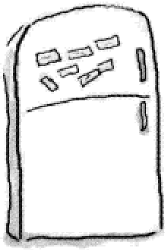
```
line_spoken = line_spoken.strip()
```

```
other = []
```

```
man.append(line_spoken)
```

```
print(man)
```

```
print(other)
```



代码磁贴答案

把代码磁贴加到现有的代码中，需要满足以下要求：

1. 创建一个空列表，名为man。
2. 再创建一个空列表，名为other。
3. 增加一行代码，删除line_spoken变量中不需要的空白符。
4. 给出条件和代码，根据role的值将line_spoken增加到适当的列表中。
5. 在屏幕上输出各个列表（man和other）。

```
man = []
other = []
try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print('The datafile is missing!')
print(man)
print(other)
```

为“man”和“other”分别赋一个空列表。

将去除空白符后的字符串再赋回到自身。

“strip()”方法从字符串中去除不想要的空白符。

根据是谁讲的话来更新其中一个列表。

“elif”表示“else if”。

最后在屏幕上显示处理后的数据。

PDG

测试驱动

将代码加载到IDLE的编辑窗口，按下F5运行。一定要把你的程序保存到sketch.txt文件所在的同一个文件夹下。

2-open-into-lists-cleaned-up.py - /Users/barryp/HeadFirstPython/chapter4/2-o...

```
man = []
other = []

try:
    data = open('sketch.txt')

    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass

    data.close()
except IOError:
    print('The datafile is missing!')

print(man)
print(other)
```

IDLE的编辑窗口。

Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
['Is this the right room for an argument?', "No you haven't!", 'When?', "No yo
u didn't!", "You didn't!", 'You did not!', 'Ah! (taking out his wallet and pay
ing) Just the five minutes.', 'You most certainly did not!', "Oh no you didn't
!", "Oh no you didn't!", "Oh look, this isn't an argument!", "No it isn't!", "
It's just contradiction!", 'It IS!', 'You just contradicted me!', 'You DID!',
'You did just then!', '(exasperated) Oh, this is futile!!', 'Yes it is!']
["I've told you once.", 'Yes I have.', 'Just now.', 'Yes I did!', "I'm telling
you, I did!", "Oh I'm sorry, is this a five minute argument, or the full half
hour?", 'Just the five minutes. Thank you.', 'Anyway, I did.', "Now let's get
one thing quite clear: I most definitely told you!", 'Oh yes I did!', 'Oh yes
I did!', 'Yes it is!', "No it isn't!", 'It is NOT!', "No I didn't!", 'No no no
!', 'Nonsense!', "No it isn't!"]
>>>
```

Ln: 8 Col: 4

这是屏幕上显示的输出：两个列表的内容。

正如所料，可以正常工作。



是的，确实可以。

如果需要把数据保存到文件，完全可以利用open() BIF。

以写模式打开文件

使用open() BIF打开磁盘文件时，可以指定使用什么访问模式。默认地，open()使用模式r表示读，所以不需要专门指定r模式。要打开一个文件完成写，需要使用模式w：

要使用的访问模式。

```
out = open("data.out", "w")
```

数据文件对象。

所写文件的文件名。

默认地，print() BIF显示数据时会使用标准输出（通常是屏幕）。要把数据写至一个文件，需要使用file参数来指定所使用的数据文件对象：

```
print("Norwegian Blues stun easily.", file=out)
```

写至文件的内容。

所写数据文件对象的名。

完成工作时，一定要关闭文件，确保所有数据都写至磁盘。这称为刷新输出（flushing），这一点非常重要：

```
out.close()
```

这在写入文件时非常非常重要。



Geek Bits

使用访问模式w时，Python会打开指定的文件来完成写。如果这个文件已经存在，则会清空它现有的内容，也就是完全清除。要追加到一个文件，需要使用访问模式a。要打开一个文件来完成写和读（不清除），需要使用w+。如果想打开一个文件完成写，但是这个文件并不存在，那么首先会为你创建这个文件，然后再打开文件进行写。

Sharpen your pencil

在程序最后，两个print() BIF会在屏幕上显示处理后的数据。下面修改这个代码，将数据保存到两个磁盘文件中。

将磁盘文件分别命名为man_data.txt(存储一个人讲的话)和other_data.txt(另一个人讲的话)。这两个数据文件打开后都要确保将其关闭，另外还要使用try/except来保护你的代码避免IOError。

```
man = []
other = []

try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print('The datafile is missing!')

........
    ..............................................
    ..............................................

    print(man, ......................)
    print(other, ......................)

    ..............................
    ..............................
........................
    ..............................
```

继续，试着写出下面的代码。

在这里打开两个数据文件。

调用"print()"时指定要写的文件。

一定要关闭这两个文件。

在这里处理出现的异常。

Sharpen your pencil Solution

在程序最后，两个print() BIF会在屏幕上显示处理后的数据。下面修改这个代码，将数据保存到两个磁盘文件中。

将磁盘文件分别命名为man_data.txt (存储一个人讲的话）和other_data.txt（另一个人讲的话）。这两个数据文件打开后都要确保将其关闭，另外还要使用try/except来保护你的代码避免IOError。

```
man = []
other = []

try:
    data = open('sketch.txt')
    for each_line in data:
        try:
            (role, line_spoken) = each_line.split(':', 1)
            line_spoken = line_spoken.strip()
            if role == 'Man':
                man.append(line_spoken)
            elif role == 'Other Man':
                other.append(line_spoken)
        except ValueError:
            pass
    data.close()
except IOError:
    print('The datafile is missing!')

try:
    man_file = open('man_data.txt', 'w')
    other_file = open('other_data.txt', 'w')

    print(man, file=man_file)
    print(other, file=other_file)

    man_file.close()
    other_file.close()
except IOError:
    print('File error.')
```

所有这些代码不做改动。

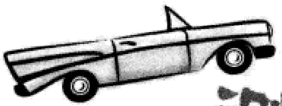
记得以写模式打开文件吧？

打开这两个文件，并分别赋至一个文件对象。

使用"print()" BIF将指定的列表保存到指定的磁盘文件。

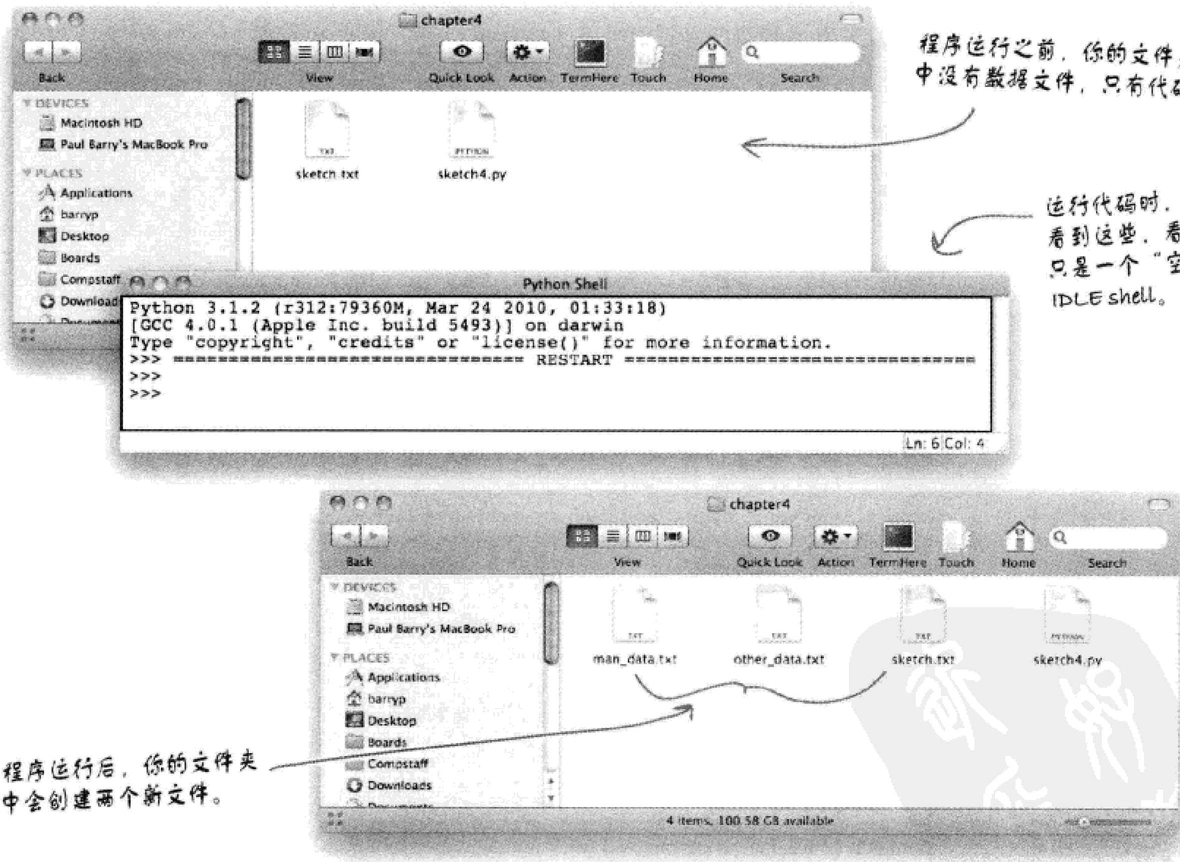
不要忘记关闭这两个文件。

如果出现I/O异常在这里处理。



测试驱动

编辑代码，将两个print()调用替换为新的文件I/O代码。然后运行程序，确认确实创建了数据文件：



这个代码也能正常工作。你创建了两个数据文件，分别包含各个列表中的数据。继续在你喜欢的编辑器中打开这两个文件，确认其中确实包含你期望的数据。

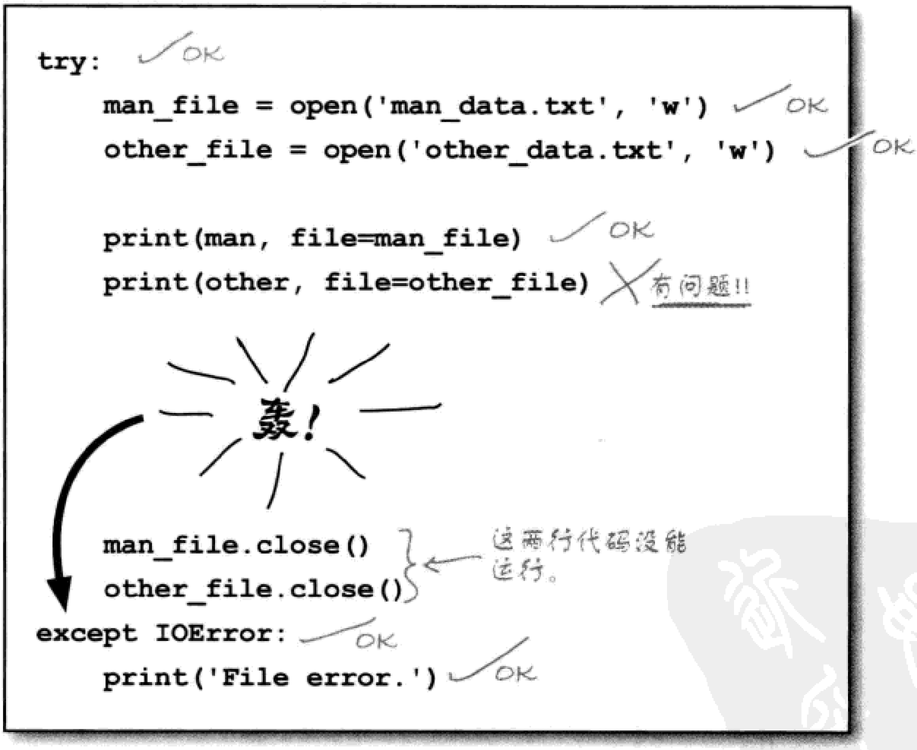
BRAIN POWER

仔细考虑下面这个问题：如果代码中的第二个print()调用导致一个IOError，你的数据文件会怎么样呢？

发生异常后文件会保持打开!

如果你要做的只是从文件读取数据，得到一个IOError确实很烦人，不过一般还不算危险，因为你的数据仍在文件中，尽管得到这些数据可能会有些麻烦。

向文件写数据就完全不同了：如果在文件关闭前需要处理一个IOError，所写的数据可能会被破坏，而且只有这种情况真正发生了才能知道，否则根本无法了解这一点。



异常处理代码完成了它的任务，不过你现在遇到了一个棘手的情况：你的数据有可能被破坏，这可不好。

这里需要一种策略，保证不论是否出现一个IOError都会运行某些代码。在代码上下文中，你希望不论发生了什么都要确保关闭文件。

用finally扩展try

如果遇到这种情况，也就是不论出现什么错误都必须运行某些代码时，可以向try语句的finally组增加代码：

这里没有变化，只不过……

```
try:
    man_file = open('man_data.txt', 'w')
    other_file = open('other_data.txt', 'w')

    print(man, file=man_file)
    print(other, file=other_file)

except IOError:
    print('File error.')

finally:
    man_file.close()
    other_file.close()
```

……要把"close()"调用移到这里来。

如果没有出现任何运行时错误，会执行finally组中的代码。同样的，如果出现IOError，会执行except组，然后运行finally组。

不论什么情况，finally组中的代码总会运行。

通过把文件关闭代码移入finally组中，可以减少数据破坏错误的可能性。

这是一个很大的改进，因为你现在可以确保文件妥善地关闭（即使出现写错误）。

不过那些错误呢？

如何发现错误的特定信息？

PDG

there are no Dumb Questions

问： **我有些糊涂了，你从line_spoken数据中去掉不想要的空白符时，把结果又赋回给line_spoken变量。看来在line_spoken上调用strip()方法就会改变它指示的字符串，是这样吗？**

答： 不，并非如此。Python中的字符串是不可变的，这说明一旦创建一个字符串，它就不会再改变了。

问： **但是你确实改变了line_spoken字符串，从中删除了你不想要的空白符，难道不是吗？**

答： 也对，也不对。实际上，在line_spoken上调用strip()方法会创建一个新的字符串，其中去除了原字符串中前面和后面的空白符。这个新字符串再赋至line_spoken，取代之前所指示的数据。事实上，尽管看上去好像改变了line_spoken，但其实只是完全替换了它指示的数据。

问： **那么被替换的数据怎么样了呢？**

答： Python内置的内存管理技术会回收所使用的RAM，以便你的程序使用。也就是说，除非还有另外某个Python数据对象也指示这个字符串，否则Python会将它回收。

问： **为什么呢？我不太明白。**

答： 另一个数据对象可能也指示line_spoken所指示的字符串，这一点是可以理解的。例如，下面假设代码中包含两个变量，它们都指向同一个字符串"Flying Circus"。然后你决定要让其中一个变量变成全大写，所以在这个对象上调用upper()方法。Python解释器会得到这个字符串的一个副本，将它转换为大写，再返回给你。然后你可以把这个转换为大写的数据赋回到指向原数据的那个变量。

问： **这么说，原来的数据不能改变，因为还有一个变量指向它，是吗？**

答： 完全正确。正是这个原因字符串是不可变的，因为你永远不会知道还有哪些变量指向某个特定的字符串。

问： **但是Python肯定能知道有多少个变量在指示某个特定的字符串，不是吗？**

答： 确实如此，不过这只是为了垃圾回收。如果有这样一行代码：print('Flying Circus')，这个字符串并未由某个变量指示（所以变量引用计数不会把它计入），不过这仍然是一个合法的字符串对象（可能会有某个变量指示它），所以在任何情况下都不能改变它的数据。

问： **这么说，尽管可以为Python变量赋数据，但实际上变量并不包含所赋的数据，是吗？**

答： 非常正确。Python变量只包含数据对象的一个引用。数据对象才真正包含数据，可以想见，一个字符串对象可能在代码中的多个不同位置使用，所以最安全的做法就是让所有字符串都不可变，以免产生不好的副作用。

问： **这样就无法"就地"调整字符串了，这不算一个很大的麻烦吗？**

答： 不，并不麻烦。一旦习惯了字符串的用法，这就不算什么问题了。实际上，你会发现这一点几乎不会对你造成障碍。

问： **还有其他Python数据类型也是不可变的吗？**

答： 是的，很多都是不可变的。比如说元组，这是一个不可变的列表。另外，所有数值类型都是不可变的。

问： **除了具体了解每种类型，我怎么才能知道一个类型是不可变的呢？**

答： 不用担心：你会知道的。如果你想改变一个不可变的值，Python会产生一个TypeError异常。

问： **当然了：会出现一个异常。异常在Python中无处不在，不是吗？**

答： 确实是，正是异常才让这个世界变幻莫测。

知道错误类型还不够

出现文件I/O错误时，你的代码会显示一个一般的“File error”消息。这太过于一般化了。你怎么知道到底发生了什么呢？



谁知道呢？

看起来Python解释器知道……只要你问它就会告诉你有关的细节。

运行时出现一个错误时，Python会产生一个特定类型的异常（如`IOError`、`ValueError`等）。另外，Python会创建一个异常对象，它会作为一个参数传入`except`代码组。

下面使用IDLE来看这是怎样做的。

An IDLE Session

下面来看试图打开一个根本不存在的文件时会发生什么，如要打开一个名为missing.txt的磁盘文件。在IDLE的shell输入以下代码：

```
>>> try:
        data = open('missing.txt')
        print(data.readline(), end='')
except IOError:
        print('File error')
finally:
        data.close()
```

```
File error
Traceback (most recent call last):
  File "<pyshell#8>", line 7, in <module>
    data.close()
NameError: name 'data' is not defined
```

这里是你的错误消息，不过……

……这是什么？产生了另一个异常，它导致你的代码崩溃。

文件不存在时，数据文件对象并未创建，这样就不可能在数据对象上调用close()方法，所以最后会得到一个NameError错误。一种简便的修正方法是对finally组增加一个简单的测试，尝试调用close()之前先查看data名是否存在。locals() BIF会返回当前作用域中定义的所有名的一个集合。下面利用这个BIF，只有在安全时才调用close()：

```
finally:
    if 'data' in locals():
        data.close()
```

"in"操作符测试成员关系。

只需对代码做这一点修改。按下Alt-P，在IDLE的shell中编辑代码。

```
File error
```

这一次没有其他异常了，只会显示你的错误消息。

这里会在locals() BIF返回的集合中搜索字符串data。如果找到，可以认为文件已经成功打开，可以安全地调用close()方法。

如果出现其他错误（代码调用print() BIF时可能有问题），你的异常处理代码会捕获到这个错误，显示“File error”消息，最后关闭所有打开的文件。

不过你还是不清楚到底是什么导致了这个错误。

产生一个异常并由except组处理时，Python解释器将一个异常对象传入这个except组。只需做一个很小的修改就可以在代码中使用这个异常（作为一个标识符）：

```
except IOError as err:
       print('File error: ' + err)
```

为异常对象给定一个名……

……然后用作为错误消息的一部分。

不过做这个修改后试图运行代码时，会生成另一个异常：

```
Traceback (most recent call last):
  File "<pyshell#18>", line 5, in <module>
    print('File error:' + err)
TypeError: Can't convert 'IOError' object to str implicitly
```

唉呀！又有一个异常，这一次是一个"TypeError"。

这一次你的错误消息根本没有出现。看来异常对象和字符串类型不兼容，所以尝试把异常对象与字符串相连接会带来问题。可以使用str() BIF把异常对象转换（或强制转换）为字符串：

```
except IOError as err:
       print('File error: ' + str(err))
```

使用"str()" BIF要求异常对象表现为一个字符串。

现在，做最后这个修改后，代码终于有了你期望的表现：

```
File error: [Errno 2] No such file or directory: 'missing.txt'
```

现在你会得到一个特定的错误消息，指出到底哪里出了问题。

当然，所有这些额外的逻辑会混淆你的代码的真正意图。



如果能有一种办法利用这些机制而不让代码膨胀，那该多好，我想这可能只是异想天开吧……

用with处理文件

由于处理文件时**try/except/finally**模式相当常用，所以Python提供了一个语句来抽象出相关的一些细节。对文件使用`with`语句时，可以大大减少需要编写的代码量，因为有了`with`语句就不再需要包含一个`finally`组来处理文件的关闭，即妥善关闭一个可能打开的数据文件。来看下面的例子：

```
try:
    data = open('its.txt', "w")
    print("It's...", file=data)
except IOError as err:
    print('File error: ' + str(err))
finally:
    if 'data' in locals():
        data.close()
```

这是通常的"try/except/finally"模式。

```
try:
    with open('its.txt', "w") as data:
        print("It's...", file=data)
except IOError as err:
    print('File error: ' + str(err))
```

使用"with"就不再需要"finally"组了。

使用`with`时，不再需要操心关闭打开的文件，因为Python解释器会自动为你考虑这一点。右边的`with`代码在功能上等同于左边的代码。在Head First Labs，我们很清楚自己更喜欢哪一种方法。

Geek Bits

with语句利用了一种名为上下文管理协议（context management protocol）的Python技术。

PDG

Sharpen your pencil

拿出笔来，使用with重写这个try/except/finally代码。以下是增加了适当finally组的代码：

```
try:
    man_file = open('man_data.txt', 'w')
    other_file = open('other_data.txt', 'w')

    print(man, file=man_file)
    print(other, file=other_file)
except IOError as err:
    print('File error: ' + str(err))
finally:
    if 'man_file' in locals():
        man_file.close()
    if 'other_file' in locals():
        other_file.close()
```

在这里写出你的"with"代码。

Sharpen your pencil Solution

拿出笔来，使用with重写这个try/except/finally代码。以下是增加了适当finally组的代码：

```
try:
    man_file = open('man_data.txt', 'w')
    other_file = open('other_data.txt', 'w')

    print(man, file=man_file)
    print(other, file=other_file)
except IOError as err:
    print('File error: ' + str(err))
finally:
    if 'man_file' in locals():
        man_file.close()
    if 'other_file' in locals():
        other_file.close()
```

```
try:
    with open('man_data.txt', 'w') as man_file:
        print(man, file=man_file)
    with open('other_data.txt', 'w') as other_file:
        print(other, file=other_file)
except IOError as err:
    print('File error: ' + str(err))
```

使用两个"with"语句重写代码，这一次不包括"finally"组。

或者将两个"open()"调用合并到一个"with"语句中。

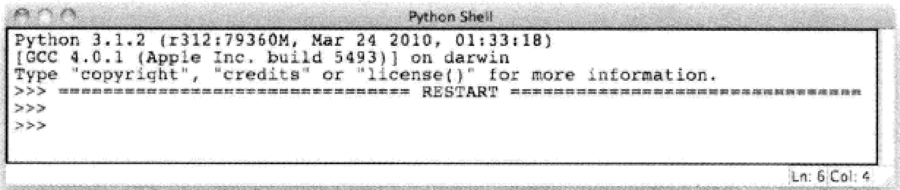
注意逗号的使用。

```
with open('man_data.txt', 'w') as man_file, open('other_data.txt', 'w') as other_file:
    print(man, file=man_file)
    print(other, file=other_file)
```

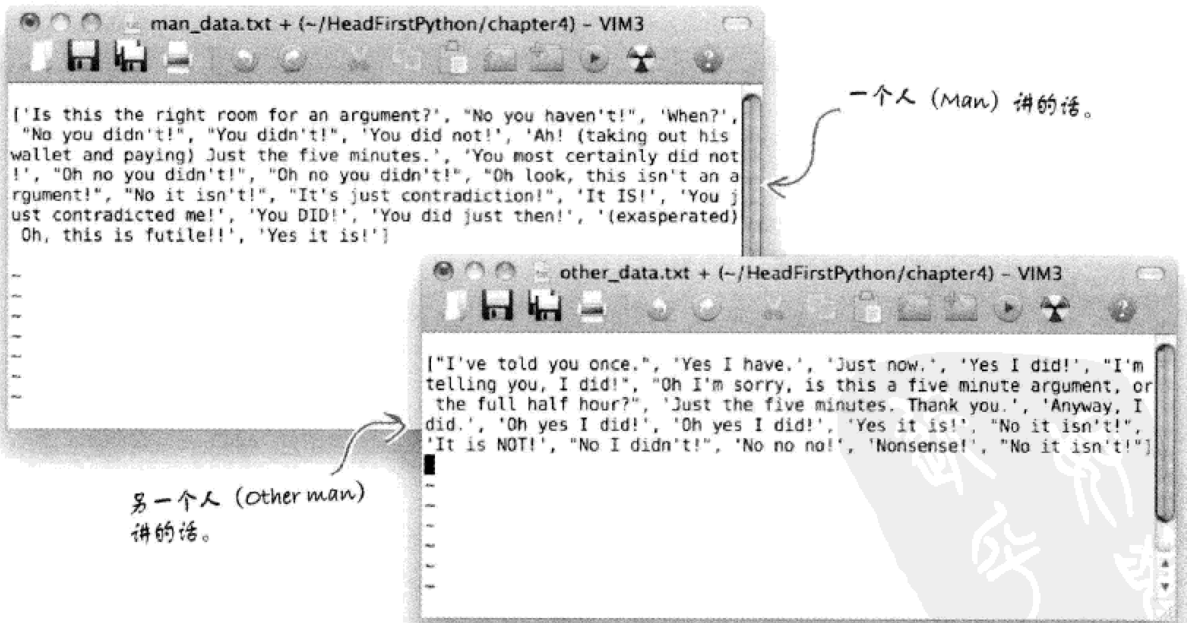
测试驱动

将with代码增加到程序中，确认它能继续正常工作。删除用前一版本程序创建的两个数据文件，然后把最新的代码加载到IDLE，试着运行这个代码。

IDLE shell中没有出现错误，这说明程序已经成功运行。



如果检查文件夹，那两个数据文件应该已经再次出现。下面用你喜欢的文本编辑器（或者使用IDLE）打开这两个数据文件，进一步查看数据文件的内容。



一个人（Man）讲的话。

另一个人（Other man）讲的话。

你已经将两个列表保存到两个文件中，其中分别包含一个人（Man）和另一个人（Other man）讲的话。你的代码会足够聪明，可以处理Python或操作系统可能抛出的任何异常。

干得漂亮。确实很有进步。

默认格式对文件并不合适

尽管现在数据已经存储在文件中，但采用的并不是一个有用的格式。下面在IDLE shell中做个试验，看看这可能会产生什么影响。

An IDLE Session

使用一个with语句打开数据文件，并显示其中的一行：

```
>>> with open('man_data.txt') as mdf:
        print(mdf.readline())
```

说明：不需要关闭文件，因为"with"会为你完成这个工作。

```
['Is this the right room for an argument?', "No you haven't!", 'When?', "No you didn't!", "You didn't!", 'You did not!', 'Ah! (taking out his wallet and paying) Just the five minutes.', 'You most certainly did not!', "Oh no you didn't!", "Oh no you didn't!", "Oh look, this isn't an argument!", "No it isn't!", "It's just contradiction!", 'It IS!', 'You just contradicted me!', 'You DID!', 'You did just then!', '(exasperated) Oh, this is futile!!', 'Yes it is!']
```

哇！看起来保存时你的列表被print()转换为一个庞大的字符串。这个试验代码从文件读入一行数据，得到所有数据，这将是一个很大的文本块……你的代码保存列表数据时所做的就是这些。

处理这个问题有哪些选择？

Geek Bits

默认地，print()会模仿Python解释器实际存储列表数据的格式来显示你的数据。所得到的输出不再做进一步处理……其主要作用只是告诉你（Python程序员）数据在内存中的"样子"。



解析文件中的数据是可以的……不过由于数据中那些中括号、引号和逗号的存在，这会很复杂。编写必要的代码当然是可行的，不过仅仅是读回所保存的数据就需要大量代码。

当然，如果数据采用一种更容易解析的格式，这个任务会容易得多，所以也许第二种选择值得考虑，是吧？

BRAIN POWER

你能想出本书前面创建的哪个函数对此有帮助？

何不修改print_lol()?

回忆第2章创建的print_lol()函数，它接收任意的列表（或者列表的列表），在屏幕上显示出来，一次显示一行。而且如果需要，嵌套列表可以缩进。

这个函数听起来相当完美！以下是nester.py模块中的代码（第2章最后见过）：

nester.py - /Users/barryp/Downloads/nester-1.3.0/nester.py

```
"""This is the "nester.py" module and it provides one function called print_lol()
   which prints lists that may or may not include nested lists."""

def print_lol(the_list, indent=False, level=0):
    """ Prints a list of (possibly) nested lists.

        This function takes a positional argument called "the_list", which
        is any Python list (of - possibly - nested lists). Each data item in the
        provided list is (recursively) printed to the screen on it's own line.
        A second argument called "indent" controls whether or not indentation is
        shown on the display. This defaults to False: set it to True to switch on.
        A third argument called "level" (which defaults to 0) is used to insert
        tab-stops when a nested list is encountered."""

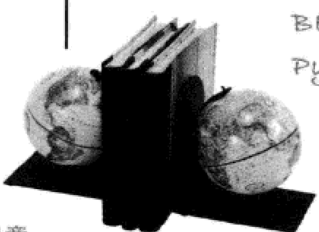
    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item, indent, level+1)
        else:
            if indent:
                for tab_stop in range(level):
                    print("\t", end='')
            print(each_item)
```

Ln: 24 Col: 0

目前这个代码在屏幕上显示数据。

修改这个代码，让它把数据输出到一个磁盘文件而不是显示在屏幕上（称为标准输出），这应该很简单。然后就可以采用一种更可用的格式保存数据。

the Scholar's Corner



标准输出（Standard Output） 这是使用"print()"BIF时代码写数据的默认位置。这通常是屏幕。在Python中，标准输出是指"sys.stdout"，可以从标准库的"sys"模块导入。



Exercise

1. 下面向print_lol()函数增加第4个参数，用来标识将把数据写入哪个位置。一定要为这个参数提供一个缺省值sys.stdout，这样如果调用这个函数时没有指定文件对象则会依然写至屏幕。

   在下面的空白处填入这个新参数的细节（注意：为了节省空间，以下代码省去了注释，不过修改代码后一定要更新nester.py模块中的注释）。

```
def print_lol(the_list, indent=False, level=0, ........................):

    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item, indent, level+1, ..........)
        else:
            if indent:
                for tab_stop in range(level):
                    print("\t", end='', ...............)
            print(each_item, ....................)
```

2. 既然已经有了修改后的print_lol()函数，with语句中的代码需要如何调整？

   ....

   ....

   ....

3. 列出程序需要导入的模块名（可能需要多个模块），来支持对print_lol()的修改。

   ....

   ....



1. 下面向print_lol()函数增加第4个参数，用来标识将把数据写入哪个位置。一定要为这个参数提供一个缺省值sys.stdout，这样如果调用这个函数时没有指定文件对象则会依然写至屏幕。

   在下面的空白处填入这个新参数的细节（注意：为了节省空间，以下代码省去了注释，不过修改代码后一定要更新nester.py模块中的注释）。

增加第4个参数，并指定缺省值

```
def print_lol(the_list, indent=False, level=0, fh=sys.stdout):

    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item, indent, level+1, fh)
        else:
            if indent:
                for tab_stop in range(level):
                    print("\t", end='', file=fh)
            print(each_item, file=fh)
```

注意：签名已经改变。

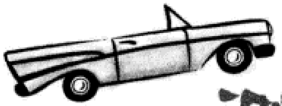
调整两个"print()"调用来使用这个新参数。

2. 既然已经有了修改后的print_lol()函数，with语句中的代码需要如何调整？

   需要调整代码，从而不再使用"print()" BIF，代码需要调用"print_lol()"。

3. 列出程序需要导入的模块名（可能需要多个模块），来支持对print_lol()的修改。

   程序需要导入修改后的"nester"模块。



测试驱动

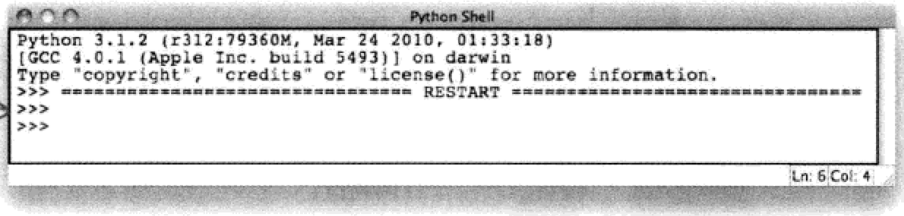
测试代码之前，需要先完成以下步骤：

1. 对nester做必要的修改，并把修改后的模块安装到你的Python环境中（可以复习第2章中的有关说明）。你可能还想上传到PyPI。
2. 修改程序，导入nester，并在with语句中使用print_lol()而不是print()。注意：你的print_lol()调用形式应当如下：

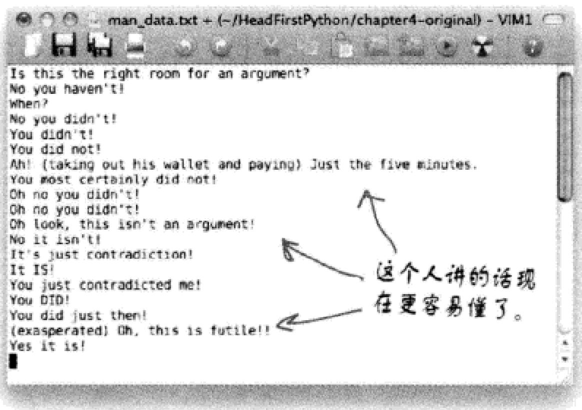
```
print_lol(man, fh=man_file)
```

一旦准备就绪，测试这个最新的程序，看看会发生什么：

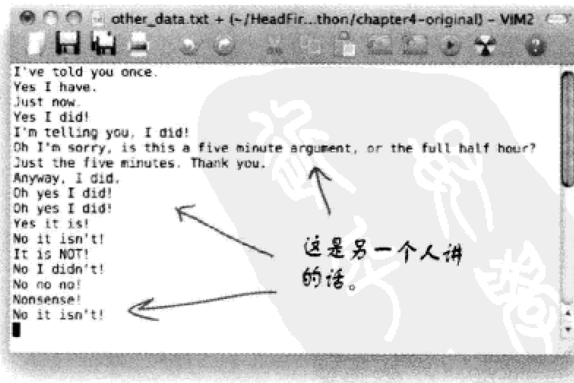
与前面一样，屏幕上没有输出。



下面检查文件的内容，看看现在内容是怎样的。



这个人讲的话现在更容易懂了。



这是另一个人讲的话。

看起来很好。通过修改nester模块，你提供了一个工具，可以采用一种易读的格式保存列表数据。现在读数据就容易多了。

不过这样能够更容易地读回数据吗？

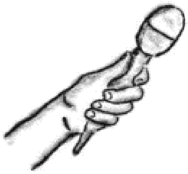


问得好。

这个问题与本章最前面的问题并没有不同，因为你已经得到一个磁盘文件中的文本行，需要进行处理，只不过现在你有两个文件而不是一个。

你知道如何编写代码来处理你的新文件，但是像这样编写定制代码会特定于你针对这个问题创建的格式。这种做法很脆弱：如果数据格式改变，你的定制代码也必须改变。

问问自己：这样值得吗？



定制代码剖析

本周访谈：
定制代码何时适用?

Head First: 你好，定制代码，今天怎么样?

定制代码： 你好，好极了！如果我感觉不太好，肯定是我得做些什么来修正一些问题。对我来说一切都轻而易举。来，请坐。

Head First: 好的，谢谢。

定制代码： 我来给你解释一下。这是我的最新定制SlideBack&Groove™，2011款，增加了靠垫和腰部支持……而且它会根据你的体形自动调整。感觉怎么样?

Head First: 实在是太舒服了[放松]，感觉真不错。

定制代码： 看到了吧？对我来说根本无难事。只要你有问题都可以来找我。你尽管开口，只要是定制任务，什么都能办到。

Head First: 这正是我来这里的原因。我有一个“小”问题想问你。

定制代码： 问吧，尽管问。我会给你满意的回答。

Head First: 定制代码何时适用?

定制代码： 这不是很明显吗？什么时候都适用！

Head First: 即使以后会带来问题?

定制代码： 问题？不过我已经告诉过你，对我来说根本无难事。我的存在就是为了完成定制。如果有问题，我当然能修正。

Head First: 即使有一个现成的方案可以做更好的修正，你也不考虑，是吗?

定制代码： 现成的？你是说（我真是懒得说）：现货?

Head First: 对。特别是在写复杂程序的时候，是吗?

定制代码： 什么？这正是我最擅长的：为遇到复杂计算问题的人创建精巧的定制解决方案。

Head First: 但是如果有人以前已经做过，为什么还要重来一次呢?

定制代码： 不过我做的一切都是定制的，这正是人们需要我的原因……

Head First: 没错，但是如果你能利用其他编码者的工作，你就能用更少的代码花一半的时间来完成你自己的工作。这一点不可否认吧，是不是?

定制代码： “利用”……听上去难道不像是“非法利用”吗?

Head First: 这更应算是协作、共享、参与和合作。

定制代码： [震惊]你想让我把我的代码……交出去?

Head First: 嗯……这更像是共享。如果你帮我，我也会帮你。听上去怎么样?

定制代码： 听起来真让人不舒服。

Head First: 你太好笑了[大笑]。我的意思只是说，既然有很好的解决方案可以解决问题，却一切都用定制代码从头开始，这通常并不是一个好的想法。

定制代码： 我同意……不过那肯定不如这个椅子这么合适。

Head First: 不过不会影响人们坐！

定制代码： [大笑] 你应该和我的朋友Pickle聊聊……他一直在专攻这样的事情。可糟糕的是，他住在库里。

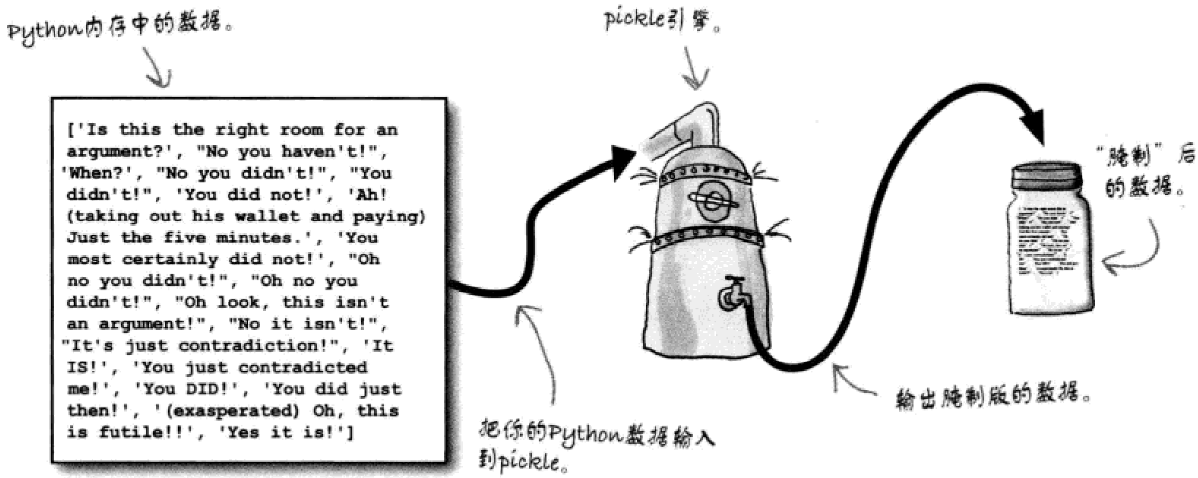
Head First: 我想我会找他的。谢谢！

定制代码： 要记住：如果你需要完成定制工作，应该知道到哪里找我。

"腌制"数据

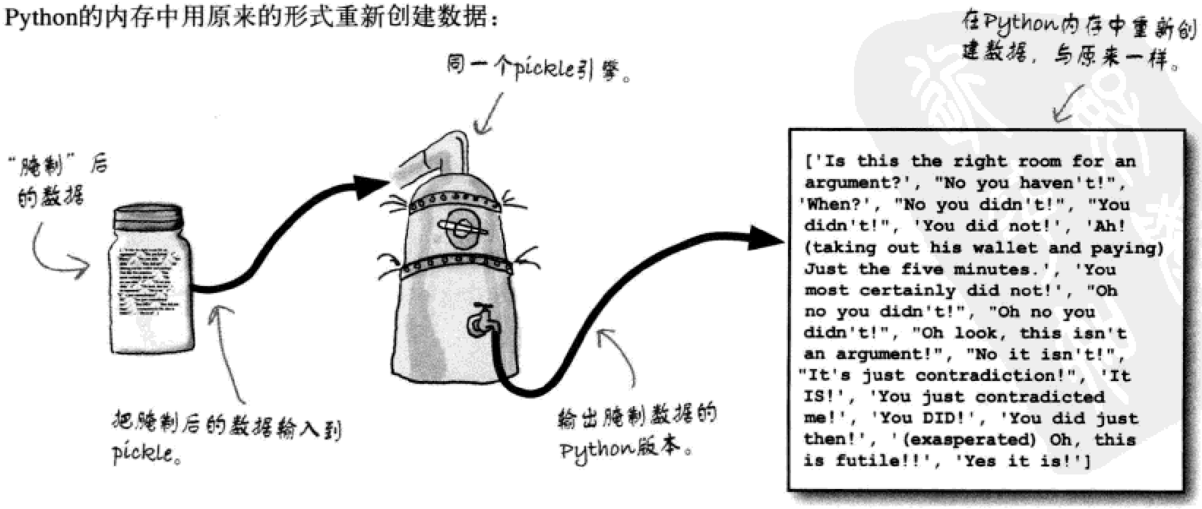
Python提供了一个标准库，名为pickle，它可以保存和加载几乎任何Python数据对象，包括列表。

一旦把数据"腌制"到一个文件，它将会持久存储，可以在以后某个日期/时间读入另外一个程序。



当然，你可以把"腌制"的数据存储在磁盘上，放在数据库中，或者通过网络传输到另一个计算机。

准备就绪后，将这个过程反过来，将持久存储的腌制数据解除腌制，在Python的内存中用原来的形式重新创建数据：



用dump保存，用load恢复

使用pickle很简单：只需导入所需的模块，然后使用dump()保存数据，以后某个时间使用load()恢复数据。处理腌制数据时的惟一要求是，必须以二进制访问模式打开这些文件：

```
import pickle
    ...
with open('mydata.pickle', 'wb') as mysavedata:
    pickle.dump([1, 2, 'three'], mysavedata)
    ...
with open('mydata.pickle', 'rb') as myrestoredata:
    a_list = pickle.load(myrestoredata)

print(a_list)
```

一定要记住导入"pickle"模块。

要保存数据，使用"dump()"。

将恢复后的数据赋至一个标识符。

"b"告诉Python以二进制模式打开数据文件。

使用"load()"从文件恢复数据。

一旦数据回到程序中，就可以像任何其他数据对象一样处理了。

如果出问题了呢？

腌制或者解除数据腌制时如果出了问题，pickle模块会产生一个PickleError类型的异常。

Sharpen your pencil

以下是现在的代码段。拿出笔来，划掉不再需要的代码，然后用使用pickle功能的代码取而代之。如果你认为有必要，还可以增加其他代码。

```
try:
    with open('man_data.txt', 'w') as man_file, open('other_data.txt', 'w') as other_file:
        nester.print_lol(man, fh=man_file)
        nester.print_lol(other, fh=other_file)
except IOError as err:
    print('File error: ' + str(err))
```

Sharpen your pencil Solution

以下是现在的代码段。拿出笔来，划掉不再需要的代码，然后用使用pickle功能的代码取而代之。如果你认为有必要，还可以增加其他代码。

在程序最前面的位置导入“pickle”。

```
import pickle

try:
    with open('man_data.txt', 'wb') as man_file, open('other_data.txt', 'wb') as other_file:
        ~~nester.print_lol(man, fh=man_file)~~      pickle.dump(man, man_file)
        ~~nester.print_lol(other, fh=other_file)~~  pickle.dump(other, other_file)
except IOError as err:
    print('File error: ' + str(err))
except pickle.PickleError as perr:
    print('Pickling error: ' + str(perr))
```

将访问模式改为“可写二进制模式”。

将两个“nester.print_lol()”调用替换为“pickle.dump()”调用。

不要忘记处理可能出现的异常。

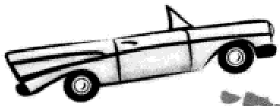
there are no Dumb Questions

问：之前调用print_lol()时，你只提供了两个参数，而函数签名要求提供4个参数。这是怎么回事？

答： 在代码中调用一个Python函数时，有很多选择，特别是函数为一些参数提供了缺省值时，如果使用位置参数，参数在函数调用中的位置会指示将哪个数据赋至哪个参数。如果函数还有一些提供了缺省值的参数，则不必操心一定为位置参数赋值。

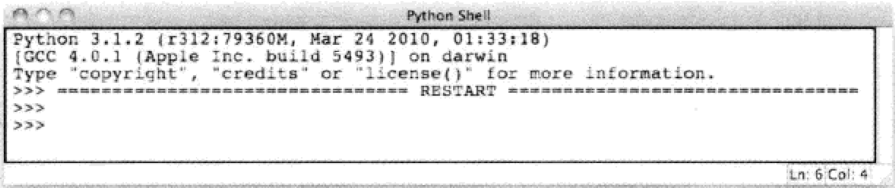
问：喂，你把我完全说糊涂了。能解释一下吗？

答： 请考虑`print()`，它的签名如下：`print(value, sep=' ', end='\n', file=sys.stdout)`。默认地，这个BIF会显示到标准输出（屏幕），因为它有一个名为`file`的参数，其缺省值为`sys.stdout`。这个`file`参数是第4个位置参数。不过，如果想把数据发送到屏幕以外的其他地方，就不需要（也不必）为第2个和第3个位置参数提供值。它们也有缺省值，所以只有在缺省值不是你想要的值时才需要为它们提供值。如果你想做的只是将数据发送到一个文件，可以这样调用print() BIF：`print("Dead Parrot Sketch", file='myfavmonty.txt')`，第4个位置参数使用你指定的值，而其他位置参数使用其缺省值。在Python中，不仅BIF采用这种方式，你的定制函数也支持这种机制。



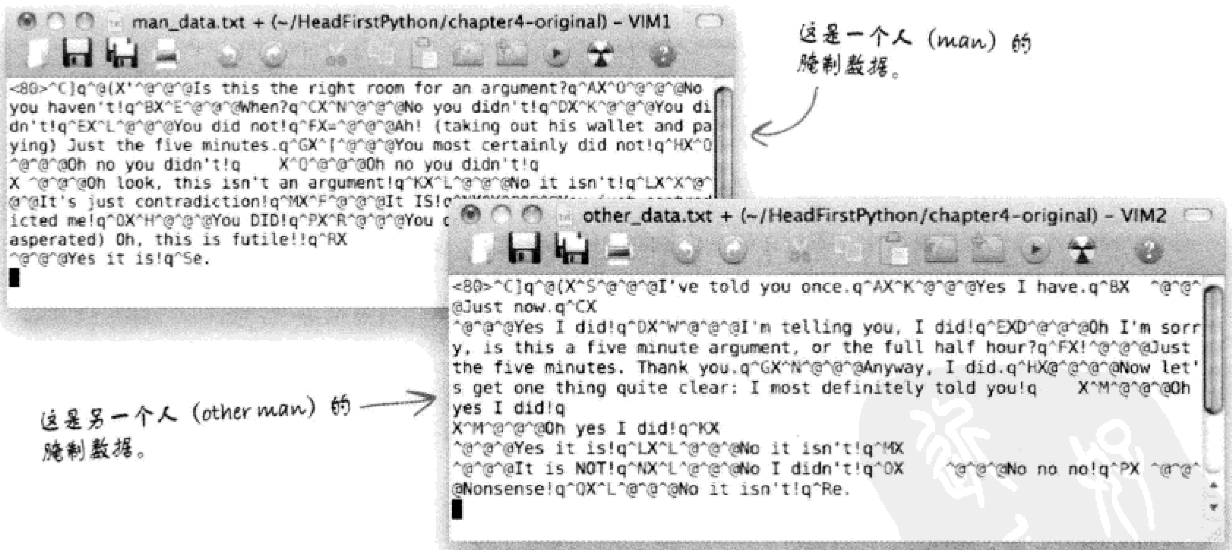
测试驱动

下面来看修改代码来使用标准pickle模块而不是你的定制nester模块后会发生什么。把修改后的代码加载到IDLE，按下F5运行代码。



同样的，这里还是没有可见的线索来指示发生的情况。

因此，再来检查文件的内容，来看现在具体内容是什么：



这是一个人（man）的腌制数据。

这是另一个人（other man）的腌制数据。

好像起作用了……不过这些文件看起来实在费解！到底怎么回事？

应该记得是Python而不是你在腌制数据。为了更高效地完成这个工作，Python的pickle模块使用了一种定制的二进制格式（这称为它的协议）。可以看到，在你的编辑器中查看这种格式看起来确实很怪异。

不用担心：它本来就是这个样子。

An IDLE Session

将原先腌制的数据加载到另一个程序时pickle表现很出色。当然，使用pickle的同时也完全可以使用nester。毕竟每个模块的设计都有各自不同的用途。下面通过IDLE shell中的一组代码来展示这一点。首先导入所有需要的模块：

```
>>> import pickle
>>> import nester
```

这里没什么奇怪的吧？

继续：创建一个新的标识符来存放你计划解除腌制的数据。创建一个空列表，名为new_man。

```
>>> new_man = []
```

没错，是不是已经兴奋得说不出话了？创建了列表之后，下面把腌制的数据加载到这个列表中。与处理外部数据文件一样，最好用try/except把你的代码包起来：

```
>>> try:
        with open('man_data.txt', 'rb') as man_file:
                new_man = pickle.load(man_file)
except IOError as err:
        print('File error: ' + str(err))
except pickle.PickleError as perr:
        print('Pickling error: ' + str(perr))
```

这个代码对你来说也不算陌生。不过，现在你的数据已经被腌制，并赋至new_man列表。下面该nester施展拳脚了：

```
>>> nester.print_lol(new_man)
Is this the right room for an argument?
No you haven't!
When?
No you didn't!
 ...
You did just then!
(exasperated) Oh, this is futile!!
Yes it is!
```

这里没有显示所有数据，不过请相信，实际输出中确实包括所有数据。

最后，显示这个人讲的第一行和最后一行：

```
>>> print(new_man[0])
Is this the right room for an argument?
>>> print(new_man[-1])
Yes it is!
```

看吧，在他讲的所有话的最后，确认了这是正确的！☺

使用pickle的通用文件I/O才是上策！



Python负责你的文件I/O细节，这样你就可以重点关注你的代码实际做什么或者需要做什么。

可以看到，利用Python，处理、保存和恢复列表中的数据已经是小菜一碟。不过，Python还支持哪些其他的数据结构呢？

下面进入第5章来了解有关内容。



你的Python工具箱

你已经读完了第4章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- “不可变类型”——Python中的一些数据类型，一旦赋值，这个值就不能再改变。
- “腌制”——将数据对象保存到一个持久存储中的过程。
- “解除腌制”——从持久存储中恢复一个已保存的数据对象的过程。

BULLET POINTS

- `strip()`方法可以从字符串去除不想要的空白符。
- `print()` BIF的`file`参数控制将数据发送/保存到哪里。
- `finally`组总会执行，而不论try/except语句中出现什么异常。
- 会向`except`组传入一个异常对象，并使用`as`关键字赋至一个标识符。
- `str()` BIF可以用来访问任何数据对象（支持串转换）的串表示。
- `locals()` BIF返回当前作用域中的变量集合。
- `in`操作符用于检查成员关系。
- “+”操作符用于字符串时将联接两个字符串，用于数字时则会将两个数相加。
- `with`语句会自动处理所有已打开文件的关闭工作，即使出现异常也不例外。`with`语句也使用`as`关键字。
- `sys.stdout`是Python中所谓的“标准输出”，可以从标准库的`sys`模块访问。
- 标准库的`pickle`模块允许你容易而高效地将Python数据对象保存到磁盘以及从磁盘恢复。
- `pickle.dump()`函数将数据保存到磁盘。
- `pickle.load()`函数从磁盘恢复数据。

5 推导数据

处理数据！



数据各式各样，有不同的大小、格式和编码。

要想有效地处理你的数据，通常需要把它处理并转换为一种常用的格式，以便高效处理、排序和存储。在这一章中，我们会研究Python的一些有助于有效处理数据的工具，让你感受到数据处理的非凡之处。好吧，翻开下一页，别让教练等久了……

Kelly教练需要你的帮助



这个教练是你的老朋友，你很愿意帮忙。他最好的一组U10选手一直在刻苦训练。每跑一个600米，Kelly就会计时并把时间记录在计算机上的一个文本文件中。总共有4个文件，分别记录James、Sarah、Julie和Mikey的时间数据。

james.txt

```
2-34,3:21,2.34,2.45,3.01,2:01,2:01,3:10,2-22
```

julie.txt

```
2.59,2.11,2:11,2:23,3-10,2-23,3:10,3.21,3-21
```

mikey.txt

```
2:22,3.01,3:01,3.02,3:02,3.02,3:22,2.49,2:38
```

sarah.txt

```
2:58,2.58,2:39,2-25,2-55,2:54,2.18,2:55,2:55
```

首先，这个教练需要一种快捷的方法能够很快了解每个选手跑得最快的3个时间。

你能帮忙吗？

动手做！

继续读本章后面的内容之前，先花点时间到Head First Python支持网站下载这4个数据文件。

Exercise

下面先从各个文件将数据读入各自的列表。编写一个小程序，处理各个文件，为各个选手的数据分别创建一个列表，并在屏幕上显示这些列表。

提示：试着按逗号分解数据，另外不要忘记去除不想要的空白符。

请在这里写你的代码。

Exercise Solution

下面先从各个文件将数据读入各自的列表。编写一个小程序，处理各个文件，为各个选手的数据分别创建一个列表，并在屏幕上显示这些列表。

```
withopen('james.txt') as jaf:          ← 打开文件。
    data = jaf.readline()               ← 读数据行。
james = data.strip().split(',')         ← 将数据转换为一个列表。
withopen('julie.txt') asjuf:
    data = juf.readline()
julie = data.strip().split(',')
withopen('mikey.txt') asmif:
    data = mif.readline()
mikey = data.strip().split(',')
withopen('sarah.txt') assaf:
    data = saf.readline()
sarah = data.strip().split(',')

print(james)
print(julie)
print(mikey)
print(sarah)
```

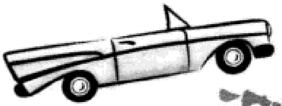
依次打开各个数据文件，从文件读取数据行，由数据行创建一个列表。（指向上面的代码块）

在屏幕上显示这4个列表。（指向4个print语句）

there are no Dumb Questions

问： 那行`data.strip().split(',')`代码看起来有些奇怪。你能解释一下这是怎么回事？

答： 这叫做方法串链（method chaining）。第一个方法`strip()`应用到`data`中的数据行，这会去除字符串中所有不想要的空白符。之后，去除了空白符的结果由第二个方法`split(',')`处理，这会创建一个列表。所得到的列表再应用到以上代码中的目标标识符。采用这种方式，可以把多个方法串链在一起，生成所需的结果。最好从左到右读这种方法链。



测试驱动

把你的代码加载到IDLE，并运行来确认现在一切正常：

这是IDLE中显示的程序。

coach.py - /Users/barryp/HeadFirstPython/chapter5/coach.py

```
with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')

print(james)
print(julie)
print(mikey)
print(sarah)
```

这是运行代码所生成的输出。

Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
['2-34', '3:21', '2.34', '2.45', '3.01', '2:01', '2:01', '3:10', '2-22']
['2.59', '2.11', '2:11', '2:23', '3-10', '2-23', '3:10', '3.21', '3-21']
['2:22', '3.01', '3:01', '3.02', '3:02', '3.02', '3:22', '2.49', '2:38']
['2:58', '2.58', '2:39', '2-25', '2-55', '2:54', '2.18', '2:55', '2:55']
>>> |
```

Ln: 10 Col: 4

到目前为止，一切都很顺利。现在Kelly教练的数据在Python内存中表示为4个列表。除了使用方法串链外，这里并没有其他新内容，因为对于如何从文件读取数据以及如何使用这些数据填充列表，你已经相当了解了。

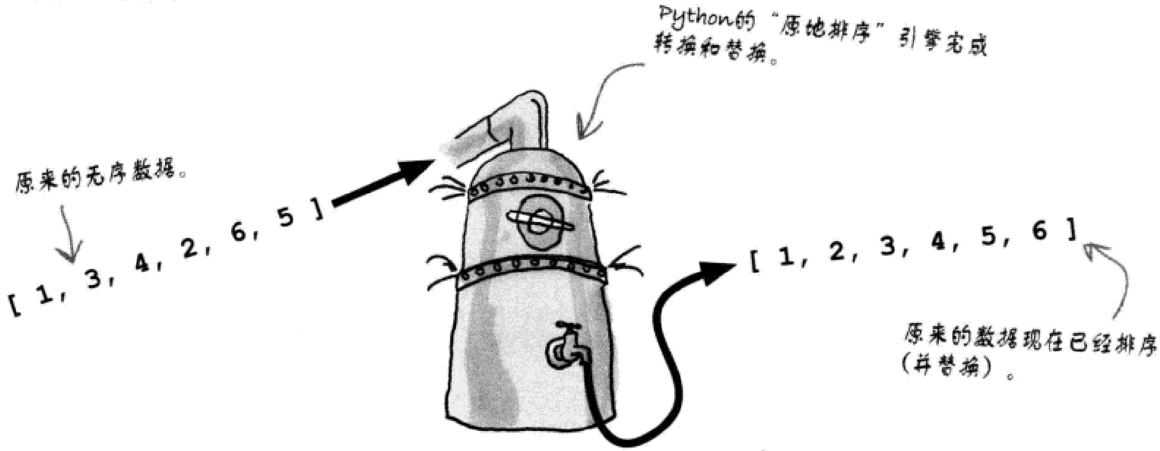
还没有得到向教练显示的内容，所以先别打扰他，下面将升序排列他的数据，这要求对数据进行排序。

下面来看Python中有哪些排序选择。

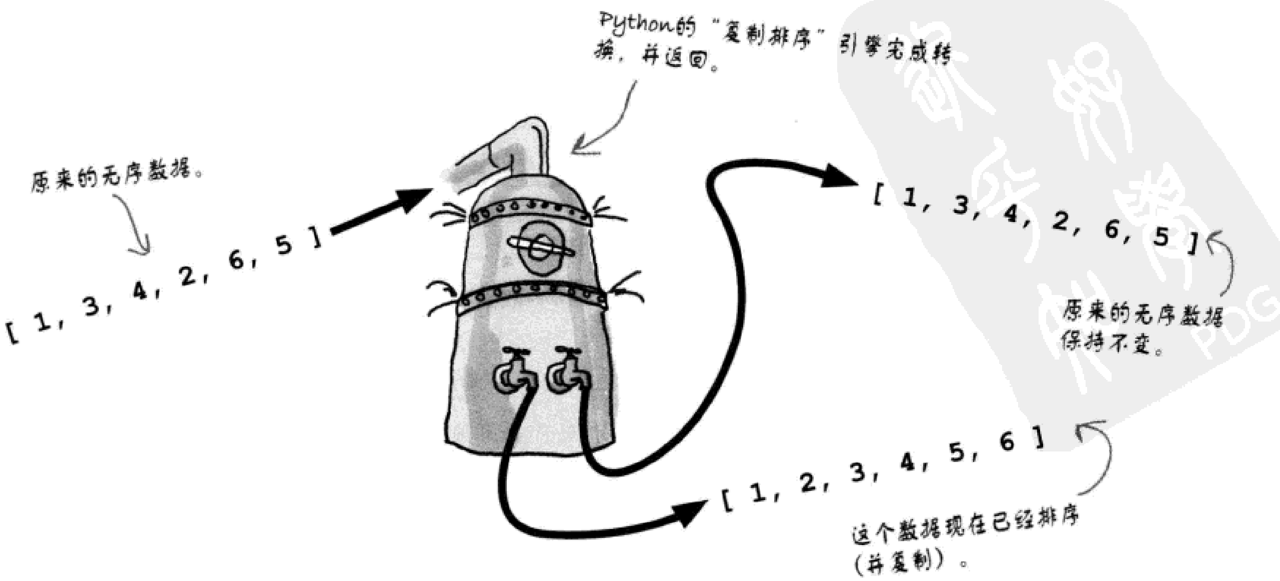
排序有两种方式

使用Python对数据排序时，你有两种选择。

原地排序（In-place sorting）是指按你指定的顺序排列数据，然后用排序后的数据替换原来的数据。原来的顺序会丢失。对于列表，`sort()`方法会提供原地排序：



复制排序（Copied sorting）是指按你指定的顺序排列数据，然后返回原数据的一个有序副本。原数据的顺序依然保留，只是对一个副本排序。在Python中，`sorted()` BIF支持复制排序。



An IDLE Session

下面来看使用Python的各个排序方法时数据有什么变化。首先在IDLE shell创建一个无序的列表：

```
>>> data = [6, 3, 1, 2, 4, 5]
>>> data
[6, 3, 1, 2, 4, 5]
```

创建一个无序数据的列表，并赋至一个变量。

使用`sort()`方法完成原地排序，这是每个Python列表都有的一个标准方法：

```
>>> data.sort()
>>> data
[1, 2, 3, 4, 5, 6]
```

`data.sort()` ← 对数据完成原地排序。

`[1, 2, 3, 4, 5, 6]` ← 数据的顺序已经改变。

将数据重置为原来的状态，然后使用`sorted()` BIF完成一个复制排序：

```
>>> data = [6, 3, 1, 2, 4, 5]
>>> data
[6, 3, 1, 2, 4, 5]
>>> data2 = sorted(data)
>>> data
[6, 3, 1, 2, 4, 5]
>>> data2
[1, 2, 3, 4, 5, 6]
```

`[6, 3, 1, 2, 4, 5]` ← 重置数据的顺序。

`data2 = sorted(data)` ← 对数据完成复制排序。

`[6, 3, 1, 2, 4, 5]` ← 与原先一样。

`[1, 2, 3, 4, 5, 6]` ← 复制的数据从小到大排序。

Sharpen your pencil

对于这个教练的数据，这两种排序选择都是可行的，不过现在我们使用复制排序对要输出的数据进行排序。请在下面的空白处给出4个修改后的`print()`语句，替换原程序最下面的4个语句。

....

....

....

....

....

....

Sharpen your pencil Solution

对于这个教练的数据，这两种排序选择都是可行的，不过现在我们使用复制排序对要输出的数据进行排序。请在下面的空白处给出4个修改后的print()语句，替换原程序最下面的4个语句。

在屏幕上显示数据之前，只需对数据调用“sorted()”。

```
print(sorted(james))
print(sorted(julie))
print(sorted(mikey))
print(sorted(sarah))
```

there are no Dumb Questions

问： 使用sort()时原来无序的数据怎么样了？

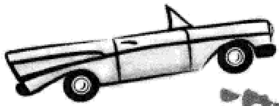
答： 总的说来，原来的数据会消失。Python会取一个副本，对它排序，然后用这个有序的版本替换原来的数据。

问： 这么说，没有办法得到原来的数据了？

答： 并非如此。如果原数据的顺序对你来说很重要，可以使用sorted() BIF将数据转换为一个有序的副本。

Geek Bits

你已经见过方法串链，现在来认识函数串链（function chaining）。函数串链允许对数据应用一系列函数。每个函数会取得数据，对它完成某个操作，然后把转换后的数据继续向下传递到下一个函数。与方法串链不同，它是从左向右读，函数串链要从右向左读（这确实很有意思）。



测试驱动

下面来看是否有办法以某种方式改进输出。对代码做必要的修改，并运行。

coach2.py - /Users/barryp/HeadFirstPython/chapter5/coach2.py

```python
with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')

print(sorted(james))
print(sorted(julie))
print(sorted(mikey))
print(sorted(sarah))
```

这是对代码的更新。

不过，请看这里！数据根本没有排序……这真有点怪异。

Python Shell

```
>>> ================================ RESTART ================================
>>>
['2-22', '2-34', '2.34', '2.45', '2:01', '2:01', '3.01', '3:10', '3:21']
['2-23', '2.11', '2.59', '2:11', '2:23', '3-10', '3-21', '3.21', '3:10']
['2.49', '2:22', '2:38', '3.01', '3.02', '3.02', '3:01', '3:02', '3:22']
['2-25', '2-55', '2.18', '2.58', '2:39', '2:54', '2:55', '2:55', '2:58']
>>> |
```

Ln: 10 Col: 4

请看这里，2-55居然在2.18前面……真奇怪。



太对了。分钟和秒的分隔符把Python的排序技术弄糊涂了。

在各个选手的文件中分别记录他们的时间时，Kelly教练使用了不同的字符来分隔分钟和秒。看起来你需要修正这些数据。

时间的麻烦

嗯……还远远不够，是不是？

仔细查看教练的数据，看看问题出在哪里。以下再次给出Sarah的原始数据：




```
2:58,2.58,2:39,2-25,2-55,2:54,2.18,2:55,2:55
```

sarah.txt

应该记得，从文件读入的数据会作为文本传入程序，所以一旦Sarah的数据转换为一个“时间”列表，数据形式如下：

```
['2:58', '2.58', '2:39', '2-25', '2-55', '2:54', '2.18', '2:55', '2:55']
```

这些都是字符串，尽管教练认为它们都是时间。

对Sarah的数据排序时，最后会得到这个顺序（这并不是你真正想要的）：

```
['2-25', '2-55', '2.18', '2.58', '2:39', '2:54', '2:55', '2:55', '2:58']
```

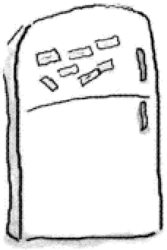
哎呀！这可不对。2.18怎么能在2-55的后面？

这里也有问题。2:39不可能在2.58和2:54之间，不是吗？



Python可以对字符串排序，对字符串排序时，短横线排在点号前面，点号则在冒号前面。所有这些字符串都以2开头，各个字符串中的下一个字符就相当于一个分组机制，有短横线的时间分组并排序，然后是包含点号的时间，最后是包含冒号的时间。

教练数据中存在的这种不一致性导致排序失败。



代码磁贴

下面创建一个函数，名为sanitize()，这个函数从各个选手的列表接收一个字符串作为输入，然后处理这个字符串，将找到的所有短横线或冒号替换为一个点号，并返回清理过的字符串。注意：如果字符串已经包含一个点号，则不需要再做清理。

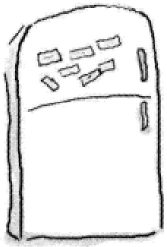
重新组织这一页下面的代码磁贴，提供所需的功能。

```
def sanitize(time_string):
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
    ........................................
    return(mins + '.' + secs)
```

向这个函数的调用者返回经过清理的时间字符串。

你的磁贴在等着你处理。

```
elif ':' in time_string:
time_string.split(splitter)
return(time_string)
if '-' in time_string:
else:
(mins, secs) =
splitter = ':'
splitter = '-'
```



代码磁贴答案

下面创建一个函数，名为sanitize()，这个函数从各个选手的列表接收一个字符串作为输入，然后处理这个字符串，将找到的所有短横线或冒号替换为一个点号，并返回清理过的字符串。注意：如果字符串已经包含一个点号，则不需要再做清理。

重新组织这一页下面的代码磁贴，提供所需的功能。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)
```

使用"in"操作符检查字符串是否包含一个短横线或冒号。

如果字符串不需要清理，就什么也不做。

分解字符串，抽出分钟和秒部分。

当然，只有sanitize()函数还不够。你还需要迭代处理每一个数据列表，使用这个新函数将各个选手的时间分别转换为正确的格式。

下面将具体使用这个新函数进行处理。

Exercise

编写代码，将现有的数据转换为经过清理的版本。创建4个新列表来保存清理后的数据。迭代处理各个选手的列表数据，从各个列表将经过清理的各个字符串追加到适当的新列表。程序的最后要在屏幕上分别打印各个新列表的一个有序副本。

从数据文件读入数据的相关代码仍保持不变（另外这里稍做压缩，以便在这一页上能够放下）。

```
with open('james.txt') as jaf: data = jaf.readline()
james = data.strip().split(',')
with open('julie.txt') as juf: data = juf.readline()
julie = data.strip().split(',')
with open('mikey.txt') as mif: data = mif.readline()
mikey = data.strip().split(',')
with open('sarah.txt') as saf: data = saf.readline()
sarah = data.strip().split(',')
```

在这里增加你的新代码。

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

这4个"print()"语句有何变化？

```
print( ............................ )
print( ............................ )
print( ............................ )
print( ............................ )
```

Exercise Solution

编写代码，将现有的数据转换为经过清理的版本。创建4个新列表来保存清理后的数据。迭代处理各个选手的列表数据，从各个列表将经过清理的各个字符串追加到适当的新列表。程序的最后要在屏幕上分别打印各个新列表的一个有序副本。

```
with open('james.txt') as jaf: data = jaf.readline()
james = data.strip().split(',')
with open('julie.txt') as juf: data = juf.readline()
julie = data.strip().split(',')
with open('mikey.txt') as mif: data = mif.readline()
mikey = data.strip().split(',')
with open('sarah.txt') as saf: data = saf.readline()
sarah = data.strip().split(',')

clean_james = []
clean_julie = []
clean_mikey = []
clean_sarah = []

for each_t in james:
    clean_james.append(sanitize(each_t))
for each_t in julie:
    clean_julie.append(sanitize(each_t))
for each_t in mikey:
    clean_mikey.append(sanitize(each_t))
for each_t in sarah:
    clean_sarah.append(sanitize(each_t))

print(sorted(clean_james))
print(sorted(clean_julie))
print(sorted(clean_mikey))
print(sorted(clean_sarah))
```

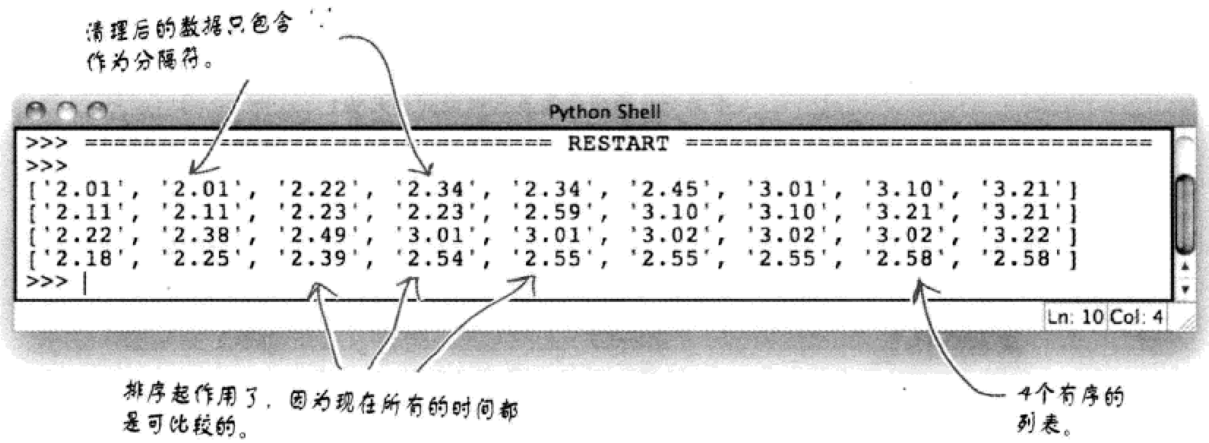
创建4个开始为空的新列表。

取原列表中的各个数据项，进行清理，然后将清理后的数据追加到适当的新列表。

这4个"print()"语句现在显示已经排序的新列表。

测试驱动

结合sanitize()函数使用上一页修改后的代码，然后在IDLE中按下F5运行代码，确认现在能正常排序。

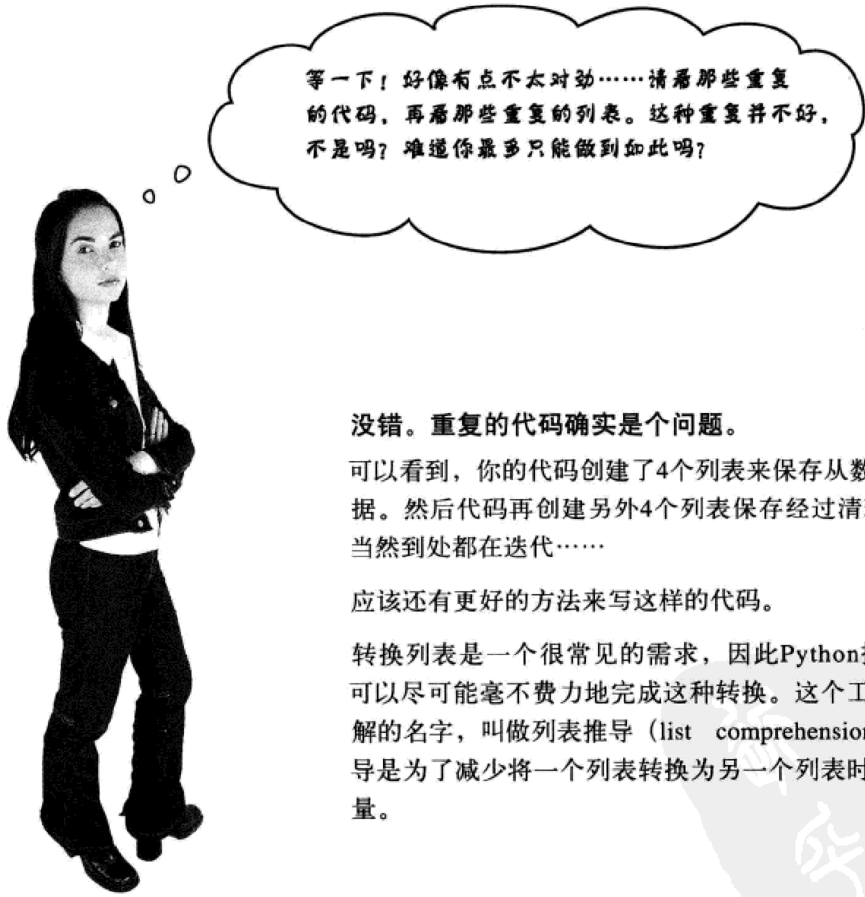


这个输出看起来好多了。

虽然得到这个结果费了一番工夫，不过现在4个文件中得到的数据不仅有序，而且格式是一致的。通过在排序前对数据进行预处理，可以有效地确保Python的排序技术正常工作。

Geek Bits

默认地，sort()方法和sorted() BIF都会按升序对数据排序。要以降序对数据排序，需要向sort()或sorted()传入参数reverse=True，Python会负责具体处理。



没错。重复的代码确实是个问题。

可以看到，你的代码创建了4个列表来保存从数据文件读取的数据。然后代码再创建另外4个列表保存经过清理的数据。另外，当然到处都在迭代……

应该还有更好的方法来写这样的代码。

转换列表是一个很常见的需求，因此Python提供了一个工具，可以尽可能毫不费力地完成这种转换。这个工具有一个相当费解的名字，叫做列表推导（list comprehension）。设计列表推导是为了减少将一个列表转换为另一个列表时所需编写的代码量。

推导列表

想想看将一个列表转换为另一个列表时需要做什么。必须做4件事，你需要：

1. 创建一个新列表来存放转换后的数据。
2. 迭代处理原列表中的各个数据项。
3. 每次迭代时完成转换。
4. 将转换后的数据追加到新列表。

```
clean_mikey = []

for each_t in mikey:
    clean_mikey.append(sanitize(each_t))
```

1.创建。
2.迭代。
3.转换。
4.追加。

下面作为列表推导完成同样的功能，这包括创建一个新列表，为此要指定将应用到一个现有列表中各个数据项的转换。

选择要使用的目标标识符（与常规的迭代类似）。

```
clean_mikey = [sanitize(each_t) for each_t in mikey]
```

创建新列表……

……为此要指定一个转换……

……应用于各个数据项……

……一个现有列表中。

有意思的是，在这里转换已经缩减为只有一行代码。另外，不再需要指定使用append()方法，因为这个动作已经隐含在列表推导中。很棒吧？

An IDLE Session

下面来看另外一些列表推导的例子。打开你的IDLE shell，完成以下单行转换。

首先将一个分钟列表转换为一个秒列表：

```
>>> mins = [1, 2, 3]
>>> secs = [m * 60 for m in mins]
>>> secs
[60, 120, 180]
```

只需将分钟值乘以60。

那么米转换成英尺呢？

```
>>> meters = [1, 10, 3]
>>> feet = [m * 3.281 for m in meters]
>>> feet
[3.281, 32.81, 9.843]
```

是的，一米等于3.281英尺。

给定一个包含混合大小写和小写字符串的列表，把它们转换成全大写的字符串简直是小菜一碟：

```
>>> lower = ["I", "don't", "like", "spam"]
>>> upper = [s.upper() for s in lower]
>>> upper
['I', "DON'T", 'LIKE', 'SPAM']
```

对每个串应用"upper()"方法。

下面使用sanitize()函数将一些列表数据转换为格式正确的时间：

```
>>> dirty = ['2-22', '2:22', '2.22']
>>> clean = [sanitize(t) for t in dirty]
>>> clean
['2.22', '2.22', '2.22']
```

可以轻松把"不干净"的东西变干净，再没有比这更容易的了☺

还可以把列表转换的结果赋回原来的目标标识符。这个例子将一个字符串列表转换为一个浮点数列表，然后替换原来的列表数据：

```
>>> clean = [float(s) for s in clean]
>>> clean
[2.22, 2.22, 2.22]
```

"float()" BIF可以转换到浮点数。

当然，如果需要，转换还可以是一个函数链：

```
>>> clean = [float(sanitize(t)) for t in ['2-22', '3:33', '4.44']]
>>> clean
[2.22, 3.33, 4.44]
```

还支持合并对数据项的转换！

Sharpen your pencil

既然认识了列表推导，下面写4个相应的代码来处理教练的4个计时值列表。将各个列表转换为经过清理的有序版本。拿出笔来，在下面给出的空白处写出你打算使用的列表推导代码。

....

....

....

....

....

....

there are no Dumb Questions

问： **那么……能不能这样考虑：列表推导好，列表迭代不好，是这样吗？**

答： 不，并不能这样看。如果必须对一个列表中的每一项完成一个转换，使用列表推导是上策，特别是如果能很容易地在一行上指定转换（或者指定为一个函数链），列表推导尤其适用。列表迭代可以完成列表推导所能完成的全部工作，只是列表迭代需要的代码更多一些，不过如果需要，迭代确实能提供更大的灵活性。

Geek Bits

Python的列表推导是这种语言支持函数编程概念的一个例子。关于开发程序代码的最佳方式有很多争论：可以是过程式编程，或者采用函数式编程技术，还可以使用面向对象技术。在Head First Labs，我们尽量不介入到这个争论当中，不过可以很高兴地告诉你：所有这3种编程方法在Python中都会以某种方式得到支持。

Sharpen your pencil Solution

既然认识了列表推导，下面写4个相应的代码来处理教练的4个计时值列表。将各个列表转换为经过清理的有序版本。拿出笔来，在下面给出的空白处写出你打算使用的列表推导代码。

列表推导会完成转换，再利用"sorted()" BIF对新列表排序。

```
sorted([sanitize(t) for t in james])

sorted([sanitize(t) for t in julie])
sorted([sanitize(t) for t in mikey])
sorted([sanitize(t) for t in sarah])
```

对其他列表重复这个工作，完成清理。

Watch it!

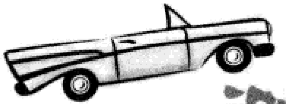
定义列表推导时，要当心在哪里使用 `sorted()` BIF。

你可能想在列表推导中使用函数链 `sorted(sanitize(t))`。千万别这么做。应该记得，一次只会对一个列表项完成转换，而不是对整个列表。在这个例子中，`sorted()` BIF希望对一个列表排序，而不是针对单个的数据项。

列表推导的妙处

通过对这个教练的选手数据使用列表推导，可以大幅减少需要你维护的代码。另外，随着你对列表推导的语法和用法越来越熟悉，你会发现列表推导的使用很自然，与你的大脑考虑数据的方式以及你希望应用的转换是一致的。

下面来确认这个新代码能够如你所愿，正常工作。



测试驱动

把前面的列表迭代代码替换为4个新的（漂亮的）列表推导代码。运行程序来确认结果没有变化。

coach4.py - /Users/barryp/HeadFirstPython/chapter5/coach4.py

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

with open('james.txt') as jaf:
    data = jaf.readline()
james
with
    d
julie
with
    d
mikey
with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')

print(sorted([sanitize(t) for t in james]))
print(sorted([sanitize(t) for t in julie]))
print(sorted([sanitize(t) for t in mikey]))
print(sorted([sanitize(t) for t in sarah]))
```

Ln: 27 Col: 0

Python Shell

```
>>> ================================ RESTART ================================
>>>
['2.01', '2.01', '2.22', '2.34', '2.34', '2.45', '3.01', '3.10', '3.21']
['2.11', '2.11', '2.23', '2.23', '2.59', '3.10', '3.10', '3.21', '3.21']
['2.22', '2.38', '2.49', '3.01', '3.01', '3.02', '3.02', '3.02', '3.22']
['2.18', '2.25', '2.39', '2.54', '2.55', '2.55', '2.55', '2.58', '2.58']
>>> |
```

Ln: 10 Col: 4

新的列表推导代码与先前的列表迭代代码生成的输出完全相同。

正如我们期望的，输出与前面完全一致。

你已经编写了一个程序，可以从Kelly教练的数据文件读入数据，将原始数据存储在列表中，将数据清理为一种一致的格式，然后排序并在屏幕上显示教练的数据。所有这些才不过25行代码。

现在可以让教练看看你的输出了。

教练会认为怎么样？



由于你匆匆清理数据并进行排序，以至于忘了考虑你实际要做什么：你的任务是生成每个选手的3次最快时间。而且，当然你的输出中不该有任何重复的时间。

访问列表中的前3个数据项很容易。可以使用标准记法指定单个的列表项，或者使用一个列表分片：

单个地访问你需要的各个数据项

```
james[0]
james[1]
james[2]

james[0:3]
```

使用一个列表分片访问列表中从第0项到（但不包括）第3项的数据项。

不过……你怎么从列表中去除重复的数据项呢？

迭代删除重复项

要处理列表来去除重复项，但在这方面列表推导帮不了你，因为重复项的删除不是一个转换，这更应算是一个过滤器。重复项删除过滤器需要在列表创建过程中检查所创建的列表，这对于列表推导来说是无法做到的。

为了满足这个新需求，你需要求助于常规的列表迭代代码。

Exercise

假设把当前程序的倒数第4行代码修改如下：

```
james = sorted([sanitize(t) for t in james])
```

也就是说，并不是在屏幕上打印清理并排序后的James数据，这行代码会把James的无序且不一致的数据替换为经过清理的有序副本。

你的下一个任务就是写一些代码从上面这行代码生成的`james`列表删除所有重复的数据项。首先创建一个新列表，名为`unique_james`，然后填入`james`中找到的唯一的（不重复的）数据项。另外，请给出代码从而只显示James的最快的3个时间。

提示：你可能会考虑使用`not in`操作符。

Exercise Solution

假设把当前程序的倒数第4行代码修改如下：

```
james = sorted([sanitize(t) for t in james])
```

也就是说，并不是在屏幕上打印清理并排序后的James数据，这行代码会把James的无序且不一致的数据替换为经过清理的有序副本。

你的下一个任务就是写一些代码从上面这行代码生成的james列表删除所有重复的数据项。首先创建一个新列表，名为unique_james，然后填入james中找到的唯一的（不重复的）数据项。另外，请给出代码从而只显示James的最快的3个时间。

提示：你可能会考虑使用not in操作符。

创建空列表，存放唯一的数据项。

```
unique_james = []
for each_t in james:
    if each_t not in unique_james:
        unique_james.append(each_t)

print(unique_james[0:3])
```

在现有数据上迭代处理……

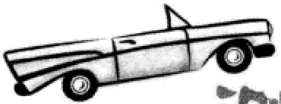
……如果这个数据项还不在新列表中……

……将这个唯一的数据项追加到新列表中。

从列表分片得到前3个数据项，并在屏幕上显示。

动手做！

对教练的其他列表（julie、mikey和sarah）重复应用这一页的以上代码。将所有新代码增加到现有程序中。



测试驱动

完成所有修改，并应用到你的程序。准备就绪后在IDLE中运行这个最新代码。

coach5.py - /Users/barryp/HeadFirstPython/chapter5/coach5.py

```
james = sorted([sanitize(t) for t in james])
julie = sorted([sanitize(t) for t in julie])
mikey = sorted([sanitize(t) for t in mikey])
sarah = sorted([sanitize(t) for t in sarah])

unique_james = []
for each_t in james:
    if each_t not in unique_james:
        unique_james.append(each_t)

print(unique_james[0:3])

unique_julie = []
for each_t in julie:
    if each_t not in unique_julie:
        unique_julie.append(each_t)

print(unique_julie[0:3])

unique_mikey = []
for each_t in mikey:
    if each_t not in unique_mikey:
        unique_mikey.append(each_t)

print(unique_mikey[0:3])

unique_sarah = []
for each_t in sarah:

pri
```

排序和清理各个列表。

删除重复的数据项。

Python Shell

```
>>> ================================ RESTART ================================
>>>
['2.01', '2.22', '2.34']
['2.11', '2.23', '2.59']
['2.22', '2.38', '2.49']
['2.18', '2.25', '2.39']
>>> |
```

Ln: 69 Col: 4

看起来不错！

成功了！

现在你只显示每个选手的最前面3个时间，而且成功地删除了重复项。

在这种情况下，列表迭代代码正是你需要的。虽然代码中有一点点重复，但还不算太糟，是吧？



这有些讽刺，但很难避免，是不是?

从列表中删除重复项的代码本身是重复的。

有时这种情况是不可避免的，有时创建一个小函数抽取出重复代码可能会有帮助。不过这里好像还有点问题……


如果有办法能够简便快速地从一个现有列表中删除重复项那该多好，不过我想这只是异想天开吧……

用集合删除重复项

除了列表，Python还提供了集合数据结构，它的表现类似于你在数学课上学到的集合。

Python中集合最突出的特性是集合中的数据项是无序的，而且不允许重复。如果试图向一个集合增加一个数据项，而该集合中已经包含有这个数据项，Python就会将其忽略。

使用set() BIF创建一个空集合，这是工厂函数的一个例子：

创建一个新的空集合，并赋至一个变量。

```
distances = set()
```

也可以一步完成集合的创建和填充。可以在大括号之间提供一个数据列表，或者指定一个现有列表作为set() BIF的参数，这就是工厂函数：

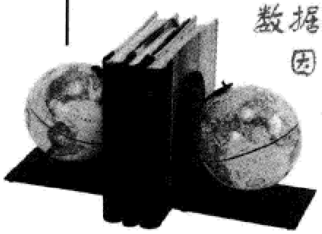
```
distances = {10.6, 11, 8, 10.6, "two", 7}
```

所提供的数据值列表中的任何重复项都会被忽略。

"james"列表中的所有重复项都被忽略。真不错。

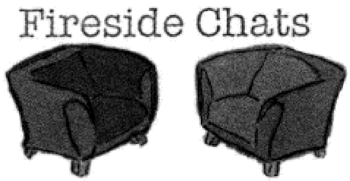
```
distances = set(james)
```

the Scholar's Corner



工厂函数：工厂函数用于创建某种类型的新的数据项。例如，"set()"就是一个工厂函数，因为它会创建一个新的集合。在真实世界中，工厂会生产产品，这个概念因此而得名。





今晚话题：集合是不是嫉妒列表？

列表： [得意地唱] “你能做的，我都能做得更好。我总是比你强。”

集合： 我不得不说，“不，你办不到。”相反，我来问你：“处理重复项呢？我看到重复项的时候，会自动把它们扔掉。”

列表： 你能拼出“d-a-t-a l-o-s-s”吗？自动丢掉数据听起来对我来说很危险。

集合： 不过这正是我要做的。集合不允许有重复的值。

列表： 真的吗？

集合： 没错。我就是因此而存在……存储值的集合。如果需要，这会非常有帮助。

列表： 你只是做这个吗？

集合： 这正是我要做的全部工作。

列表： 难道他们就为这个付钱给你？

集合： 真可笑。你一直在想方设法回避一个事实，这就是你自己根本无法去除重复的项。

列表： 你想过没有，我喜欢我的重复值。要知道，我非常爱它们。

集合： 对，也许吧。但是当你不需要它们时可就不这么想了。

列表： 这种情况很少见。而且，不管怎么说，我总能依靠其他人的帮助帮我去除那些我不需要的重复项。

集合： 我想你的意思是“set()的帮助”，是不是？

动手做！ →

> 为了抽取出你需要的数据，把当前程序中的所有列表迭代代码替换为4个`sorted(set(…))[0:3]`调用。

Head First 代码审查

Head First代码审查小组拿到你的代码，并用一种只有他们才懂的方式加了注解：他们在你的代码上勾勾划划。有些注释可以确认你已经知道的东西。另外一些是能够让你的代码变得更好的建议。像所有代码审查一样，这些注释都是为了改善代码的质量。

我想这里可以做一些改进。



来认识一下，这是我们的Head First代码审查小组。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)
```

这里最好有个注释。

如果其中某个文件找不到会怎么样？你的异常处理代码在哪里？

```
with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')
```

这里有一些重复代码。可以把代码抽取到一个小函数中。然后，你要做的就是对各个选手数据文件调用这个函数，将结果赋至一个选手列表。

```
print(sorted(set([sanitize(t) for t in james]))[0:3])
print(sorted(set([sanitize(t) for t in julie]))[0:3])
print(sorted(set([sanitize(t) for t in mikey]))[0:3])
print(sorted(set([sanitize(t) for t in sarah]))[0:3])
```

这里做了很多工作，不过我们发现如果从里向外读代码应该不难理解。

哈，OK。我们明白了。对"sorted()"生成的列表应用分片，对吗？

Exercise

下面花点时间实现审查小组的建议，将4个with语句改为一个函数。以下再列出这个代码。请在下面创建一个函数，抽出必要的功能，然后提供一个例子来说明如何在代码中调用这个新函数：

```
with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')
```

在这里写你的新函数。

给出一个示例调用。

Exercise Solution

下面花点时间实现审查小组的建议，将4个with语句改为一个函数。请在下面的空白处创建一个函数，抽出必要的功能，然后提供一个例子来说明如何在代码中调用这个新函数：

```
with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
sarah = data.strip().split(',')
```

```
def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

sarah = get_coach_data('sarah.txt')
```

接受一个文件名作为唯一的参数。

创建一个新函数。

打开这个文件，读取数据。

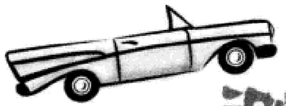
增加审查小组建议的异常处理代码。

将数据返回到代码之前先对数据完成分解/去除空白符处理。

通知你的用户有错误（如果出现错误），并返回"None"来指示失败。

调用这个函数很简单。

提供要处理的文件的名。



测试驱动

现在最后一次运行这个程序，确认使用集合可以得到列表迭代代码同样的结果。在IDLE中运行你的代码，看看会发生什么。

coach5.py - /Users/barryp/HeadFirstPython/chapter5/coach5.py

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

with open('james.txt') as jaf:
    data = jaf.readline()
james = data.strip().split(',')
```

Python Shell

```
>>> ================================ RESTART ================================
>>>
['2.01', '2.22', '2.34']
['2.11', '2.23', '2.59']
['2.22', '2.38', '2.49']
['2.18', '2.25', '2.39']
>>> |
```

Ln: 69 Col: 4

```
sarah = data.strip().split(',')

print(sorted(set([sanitize(t) for t in james]))[0:3])
print(sorted(set([sanitize(t) for t in julie]))[0:3])
print(sorted(set([sanitize(t) for t in mikey]))[0:3])
print(sorted(set([sanitize(t) for t in sarah]))[0:3])
```

Ln: 32 Col: 0

不出所料，最后这个代码很好地完成了工作。看起来不错！

太棒了！

你已经完美地处理了教练的数据，而且利用了sorted() BIF、集合和列表推导。想想看，这些技术可以应用到很多不同场合。你已经逐渐成长为一个Python数据处理专家了！





你的Python工具箱

你已经读完了第5章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- "原地"排序——转换然后替换。
- "复制"排序——转换然后返回。
- "方法串链"——从左向右读，对数据应用一组方法。
- "函数串链"——从右向左读，对数据应用一组函数。

更多Python术语

- "列表推导"——在一行上指定一个转换（而不是使用迭代）。
- "分片"——从一个列表访问多个列表项。
- "集合"——一组无序的数据项，其中不包含重复项。

BULLET POINTS

- sort()方法可以在原地改变列表的顺序。
- sorted() BIF通过提供复制排序可以对几乎任何数据结构排序。
- 向sort()或sorted()传入reverse=True可以按降序排列数据。
- 如果有以下代码：

```
new_l = []
for t in old_l:
    new_l.append(len(t))
```

使用列表推导重写这个代码，可以写作：

```
new_l = [len(t) for t in old_l]
```

- 要访问一个列表中的多个数据项，可以使用分片。例如：

```
my _ list[3:6]
```

- 这会访问列表中从索引位置3直到（但不包括）索引位置6的列表项。
- 使用set()工厂方法可以创建一个集合。

6 定制数据对象

打包代码与数据



你选择的数据结构要与数据匹配，这很重要。

而且这个选择将对代码的复杂性带来很大差别。在Python中，尽管列表和集合确实很有用，但这并不是全部。Python还提供了字典，允许你有效地组织数据，可以将数据与名关联而不是与数字关联，从而实现快速查找。当Python的内置数据结构无法胜任时，Python class语句还允许你定义自己的数据结构。这一章就会介绍这些内容。

Kelly又来了（带来一种新的文件格式）



第5章最后一个程序的输出正是这个教练想要的结果，但还有一个缺陷：从这个输出无法看出哪个数据属于哪个选手。Kelly教练认为他有办法，他已经向各个数据文件增加了标识数据：

这是增加了额外数据的“sarah2.txt”。

```
Sarah Sweeney,2002-6-17,2:58,2.58,2:39,2-25,2-55,2:54,2.18,2:55,2:55,2:22,2-21,2.22
```

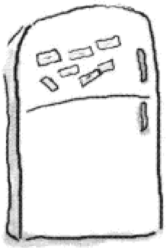
Sarah的全名　　Sarah的出生日期　　Sarah的计时数据

如果使用`split()` BIF把Sarah的数据抽取到一个列表中，第一个数据项将是Sarah的名字，第二个是她的出生日期，后面是Sarah的计时数据。

下面来分析这种格式，看看如何处理。

动手做！

请从Head First Python网站下载更新的文件。



代码磁贴

针对上一页最后给出的策略，下面来看实现这个策略的代码。对现在来说，我们主要考虑Sarah的数据。重新摆放本页最下面的代码磁贴来实现所需的列表处理，从Kelly教练的原始数据抽取并处理Sarah的3个最快时间。

提示：pop()方法从指定的列表位置删除并返回一个数据项。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)
```

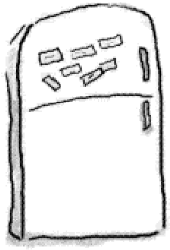
"sanitize()"函数与第5章中完全一样。

```
def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)
```

"get_coach_data()"也是从上一章得来的。

在这里重新摆放磁贴。

```
sarah
"'s fastest times are: " +
(sarah_name, sarah_dob)
=
print(sarah_name +
get_coach_data('sarah2.txt')
=
sarah.pop(0), sarah.pop(0)
str(sorted(set([sanitize(t) for t in sarah]))[0:3]))
```



代码磁贴答案

针对上一页最后给出的策略，下面来看实现这个策略的代码。对现在来说，我们主要考虑Sarah的数据。

重新摆放本页最下面的代码磁贴来实现所需的列表处理，从Kelly教练的原始数据抽取并处理Sarah的3个最快时间。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)
```

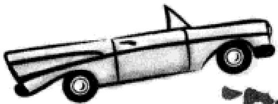
使用这个函数将Sarah的数据文件转换为一个列表，然后把它赋给"sarah"变量。

```
sarah = get_coach_data('sarah2.txt')
(sarah_name, sarah_dob) = sarah.pop(0), sarah.pop(0)
print(sarah_name + "'s fastest times are: " +
      str(sorted(set([sanitize(t) for t in sarah]))[0:3]))
```

"pop(0)"调用将删除并返回列表最前面的数据项。两个"pop(0)"调用则会删除前两个数据值，并把它们赋给指定的变量。

"print()"调用中的定制消息用来显示你想得到的结果。

PDG



测试驱动

下面在IDLE中运行这个代码，看看会发生什么。

coach2.py - /Users/barryp/HeadFirstPython/chapter6/coach2.py

```python
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

sarah = get_coach_data('sarah2.txt')

(sarah_name, sarah_dob) = sarah.pop(0), sarah.pop(0)

print(sarah_name + "'s fastest times are: " +
      str(sorted(set([sanitize(t) for t in sarah]))[0:3]))
```

最新的代码。

这个输出更容易理解。

Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>> |
```

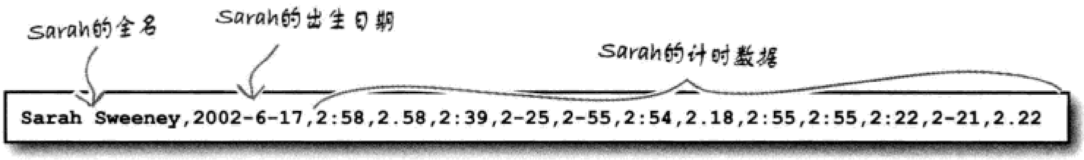
Ln: 7 Col: 4

这个程序确实能如你期望的那样正常工作，这很好……只不过你必须指定并创建Sarah的3个变量，以便标识哪个名字、出生日期和计时数据与Sarah关联。如果再增加代码来处理James、Julie和Mikey的数据，这就需要用到多达12个变量。现在这还只是在处理4个选手。不过如果要处理40、400或者4000个选手呢？

尽管在“现实生活”中数据是关联的，但在代码中一切却都是“支离破碎的”，因为表示Sarah的3部分相关数据分别存储在3个不同的变量中。

使用字典关联数据

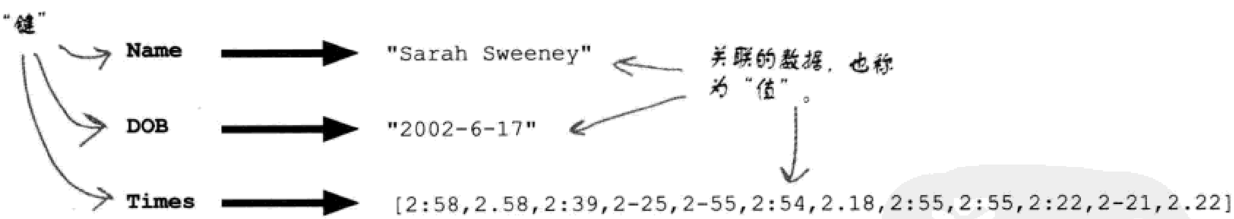
列表很不错，不过它并不能作为所有情况的最佳数据结构。下面再来看看Sarah的数据：



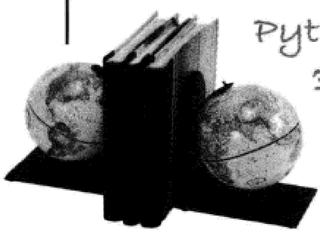
这里有一个明确的结构：选手的名字、出生日期和时间列表。

计时数据继续使用一个列表，因为这仍然很合理。不过，下面把计时数据作为另一个数据结构的一部分，利用这个数据结构，将把一个选手的所有数据与单个的变量关联。

我们将使用一个Python字典，将数据值与键关联：



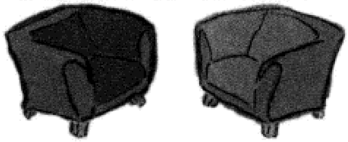
the Scholar's Corner



字典 这是一个内置的数据结构（内置于Python中），允许将数据与键而不是数字关联。这样可以使内存中的数据与实际数据的结构保持一致。



Fireside Chats



今晚话题：使用列表还是不使用列表？

字典： 嘿，你好，列表。我听说你很棒，不过不一定是复杂数据的最佳选择。这方面要看我的。

列表： 什么？你没听说吗？任何东西都可以放在列表里，不论是什么！

字典： 没错。不过如果放在列表中，就会丢失与所处理的数据关联的结构。

列表： 嗯……当然，不过前提是这个结构对你来说很重要。

字典： 这个结构当然很重要，难道不是吗？

列表： 嗯……在我看来不一定。

字典： 在你看来？在代码中对数据建模时，最好不要主观臆断。要脚踏实地，坚定不移，充满自信。所以应该使用一个字典。

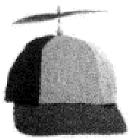
列表： 听起来像是那些自立会的口号。你是从那里听来的吗？

字典： [大笑]噢，列表，我确实很欣赏你的幽默，即使已经知道情况危急还是那么从容。要知道，道理很简单：如果你的数据有结构，就要使用字典，而不是列表。这很难吗？

列表： 确实不太难。当然，除非你是一个列表，希望对程序中每一个数据都使用列表……

字典： 但这不太可能。是否知道何时使用列表而何时使用字典，这正是从好的程序员中区分出优秀程序员的一个标准，对不对？

列表： 我想是这样。尽管你是对的，但我真的很不喜欢！



Geek Bits

Python字典在其他编程语言中还有不同的名字。如果你听到其他程序员在谈论一个“映射”、“散列”或者“关联数组”，实际上就是指“字典”。

An IDLE Session

下面来看Python字典的具体使用。在你的计算机上完成以下IDLE会话，确保能得到与这里所示相同的结果。

首先创建两个空字典，一个使用大括号创建，另一个使用一个工厂函数创建：

```
>>> cleese = {}
>>> palin = dict()
>>> type(cleese)
<class 'dict'>
>>> type(palin)
<class 'dict'>
```

不出所料，这两种技术都能创建空字典。

通过将值与键关联，向这两个字典分别增加一些数据。注意这里会自行表示出数据的具体结构，因为每个字典都有一个Name和一个Occupations列表。还要注意palin字典是一次性同时创建的（而不是分步增加数据）：

```
>>> cleese['Name'] = 'John Cleese'
>>> cleese['Occupations'] = ['actor', 'comedian', 'writer', 'film producer']
>>> palin = {'Name': 'Michael Palin', 'Occupations': ['comedian', 'actor', 'writer', 'tv']}
```

数据与键关联后（在这里，键是字符串），可以使用一种类似于列表的记法访问单个数据项：

```
>>> palin['Name']
'Michael Palin'
>>> cleese['Occupations'][-1]
'film producer'
```

使用中括号指定字典中的索引来访问数据项，不过这里中括号内不是数字，而要用键作为索引。

使用数字来访问存储在一个特定字典键位置上的列表项。可以把这认为是"索引串链"，从右向左读作："……与Occupations关联的列表的最后一项……"。译者注

与列表一样，Python字典可以动态扩展来存储额外的键/值对。下面向各个字典增加一些有关出生地的数据：

```
>>> palin['Birthplace'] = "Broomhill, Sheffield, England"
>>> cleese['Birthplace'] = "Weston-super-Mare, North Somerset, England"
```

提供与新键关联的数据。

与列表不同，Python字典不会维持插入的顺序，这会导致一些意外的行为。关于字典，重点是它会维护关联关系，而不是顺序：

```
>>> palin
{'Birthplace': 'Broomhill, Sheffield, England', 'Name': 'Michael Palin', 'Occupations':
['comedian', 'actor', 'writer', 'tv']}
>>> cleese
{'Birthplace': 'Weston-super-Mare, North Somerset, England', 'Name': 'John Cleese',
'Occupations': ['actor', 'comedian', 'writer', 'film producer']}
```

Python维护的顺序与数据插入的顺序不同。不用担心，这并没有问题。

译者注：由于英语中修饰定语放在被修饰词的后面，所以英语中确实是从右向左读，而译为中文后不能保证这一点。

Exercise

下面在代码中具体应用你现在掌握的有关Python字典的知识。现在还是重点考虑Sarah的数据。划掉你不需要的代码，换成使用字典的新代码来保存和处理Sarah的数据。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)


def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)


sarah = get_coach_data('sarah2.txt')
(sarah_name, sarah_dob) = sarah.pop(0), sarah.pop(0)
print(sarah_name + "'s fastest times are: " +
        str(sorted(set([sanitize(t) for t in sarah]))[0:3]))
```

划掉不再需要的代码。

在这里增加使用和处理字典的代码。

Exercise Solution

下面在代码中具体应用你现在掌握的有关Python字典的知识。现在还是重点考虑Sarah的数据。划掉你不需要的代码，换成使用字典的新代码来保存和处理Sarah的数据。

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)


def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

sarah = get_coach_data('sarah2.txt')
```

~~(sarah_name, sarah_dob) = sarah.pop(0), sarah.pop(0)~~
~~print(sarah_name + "'s fastest times are: " +~~
~~str(sorted(set([sanitize(t) for t in sarah]))[0:3]))~~

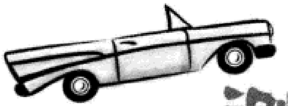
不再需要这些代码。

创建一个空字典。

```
sarah_data = {}
sarah_data['Name'] = sarah.pop(0)
sarah_data['DOB'] = sarah.pop(0)
sarah_data['Times'] = sarah
print(sarah_data['Name'] + "'s fastest times are: " +
      str(sorted(set([sanitize(t) for t in sarah_data['Times']]))[0:3]))
```

通过文件数据与字典键关联来填充字典。

处理数据时引用字典。



测试驱动

下面确认代码的这个新版本可以像前面一样正常工作，为此在IDLE环境中测试这个代码：

coach3.py - /Users/barryp/HeadFirstPython/chapter6/coach3.py

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

sarah = get_coach_data('sarah2.txt')

sarah_data = {}
sarah_data['Name'] = sarah.pop(0)
sarah_data['DOB'] = sarah.pop(0)
sarah_data['Times'] = sarah

print(sarah_data['Name'] + "'s fastest times are: " +
        str(sorted(set([sanitize(t) for t in sarah_data['Times']]))[0:3]))
```

使用字典的代码可以生成与前面完全相同的结果。

Python Shell

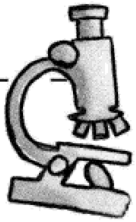
```
>>> ================================ RESTART ================================
>>>
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>> |
```

Ln: 43 Col: 4

这也能正常工作……区别在于，现在你能更容易地确定和控制哪些标识数据与哪些计时数据关联，因为它们都存储在一个字典中。

不过，说实在的，这确实需要更多代码，相应地会增加一点负担。有时额外的代码是值得的，有时也可能并不划算。不过对于这种情况通常都是“物有所值”。

下面来审查你的代码，看看能不能有所改进。



Head First 代码审查

Head First代码审查小组又在审查你的代码：他们在你的代码上勾勾划划。有些注释只是确认，另外一些则是建议。与所有代码审查一样，这些注释都是为了改善你的代码的质量。




```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        return(data.strip().split(','))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

sarah = get_coach_data('sarah2.txt')

sarah_data = {}
sarah_data['Name'] = sarah.pop(0)
sarah_data['DOB'] = sarah.pop(0)
sarah_data['Times'] = sarah

print(sarah_data['Name'] + "'s fastest times are: " +
      str(sorted(set([sanitize(t) for t in sarah_data['Times']]))[0:3]))
```

你先读入数据后再创建字典，为什么不一次完成字典的创建呢？实际上，在这种情况下，完全可以把这个处理放在get_coach_data()函数内部，让函数返回一个已经填充的字典，而不是一个列表。

这样一来，你要做的就是使用一个适当的函数调用从数据文件创建字典，对不对？

你可能想把这个代码也移到get_coach_data()函数中，因为这样做可以更好地抽出这些处理细节。不过是否这么做完全取决于你。毕竟，这是你的代码！

Sharpen your pencil

实际上，这些审查注释确实很有用。下面花点时间在你的代码中应用这些建议。这里有4个建议，你需要调整代码来支持这些建议：

1. 一次性完成字典的创建。

2. 把字典创建代码移到get_coach_data()函数中，返回一个字典而不是列表。

3. 把确定各个选手的3个最快时间的代码移到get_coach_data()函数中。

4. 调整主代码中的函数调用，调用这个新版本的get_coach_data()函数来支持新的操作模式。

拿出笔来，在下面给出的空白处编写get_coach_data()函数。给出4个函数调用来处理各个选手的数据，并提供修改后的4个print()语句：

Sharpen your pencil Solution

下面花点时间在你的代码中应用这些建议。这里有4个建议，你需要调整代码来支持这些建议：

1. 一次性完成字典的创建。

2. 把字典创建代码移到get_coach_data()函数中，返回一个字典而不是列表。

3. 把确定各个选手的3个最快时间的代码移到get_coach_data()函数中。

4. 调整主代码中的函数调用，调用这个新版本的get_coach_data()函数来支持新的操作模式。

拿出笔来，在下面给出的空白处编写get_coach_data()函数。给出4个函数调用来处理各个选手的数据，并提供修改后的4个print()语句：

```
def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return({'Name' : templ.pop(0),
                'DOB'  : templ.pop(0),
                'Times' : str(sorted(set([sanitize(t) for t in templ]))[0:3])})
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')

print(james['Name'] + "'s fastest times are: " + james['Times'])
```

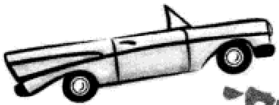
1. 在一次性创建字典之前创建一个临时列表存放数据。

2. 字典创建代码现在成为函数的一部分。

3. 确定前3个最快时间的代码也成为函数的一部分。

这里只给出了对应一个选手的这两行代码（因为可以很容易地对其他3个选手重复这些代码）。

4. 对一个选手调用函数，并根据需要调整“print()”语句。



测试驱动

下面来确认Head First代码审查小组的所有重构建议确实得到实施，并能正常完成工作。把你的代码加载到IDLE，尝试执行这些代码。

coach3c.py - /Users/barryp/HeadFirstPython/chapter6/coach3c.py

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return({'Name' : templ.pop(0),
                'DOB'  : templ.pop(0),
                'Times': str(sorted(set([sanitize(t) for t in templ]))[0:3])})
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')
julie = get_coach_data('julie2.txt')
mikey = get_coach_data('mikey2.txt')
sarah = get_coach_data('sarah2.txt')

print(james['Name'] + "'s fastest times are: " + james['Times'])
print(julie['Name'] + "'s fastest times are: " + julie['Times'])
print(mikey['Name'] + "'s fastest times are: " + mikey['Times'])
print(sarah['Name'] + "'s fastest times are: " + sarah['Times'])
```

所有数据处理都移到函数中。

这段代码已经大幅精简，现在会显示与时间关联的各个选手的名字。

Python Shell

```
>>> ================================ RESTART ================================
>>>
James Lee's fastest times are: ['2.01', '2.16', '2.22']
Julie Jones's fastest times are: ['2.11', '2.23', '2.59']
Mikey McManus's fastest times are: ['2.22', '2.31', '2.38']
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>> |
```

Ln: 19 Col: 4

看上去不错！

要处理其他选手，只需再加两行代码：第一行代码调用get_coach_data()函数，第二行代码调用print()。

如果还需要其他功能，可以编写更多函数来提供所需的功能，这并不难，对不对？



把代码与数据放在一起是对的。

要尽量将函数与函数所要处理的数据相关联，这确实很有道理，是不是？毕竟，函数只有在与数据关联时才有意义——也就是说，函数将特定于这些数据，而不是一般化的通用函数。正因如此，最好尽量把代码与相应数据打包在一起。

不过怎么做到呢？有没有一种简便的方法将定制代码（采用函数的形式）与你的定制数据相关联？

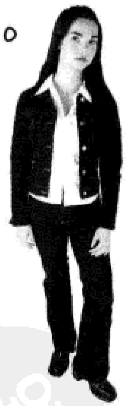
将代码及其数据打包在类中

与大多数其他现代编程语言一样，Python允许创建并定义面向对象的类，类可以用来将代码与代码处理的数据相关联。



使用类有助于降低复杂性。

通过将代码与代码处理的数据相关联，随着代码基的扩大，可以降低复杂性。



降低复杂性意味着bug更少。

降低复杂性会使你的代码中bug更少。不过，有一点需要知道，这就是经过一段时间后你的程序可能会增加功能，而这可能带来额外的复杂性。使用类来管理这种复杂性是一种非常好的做法。



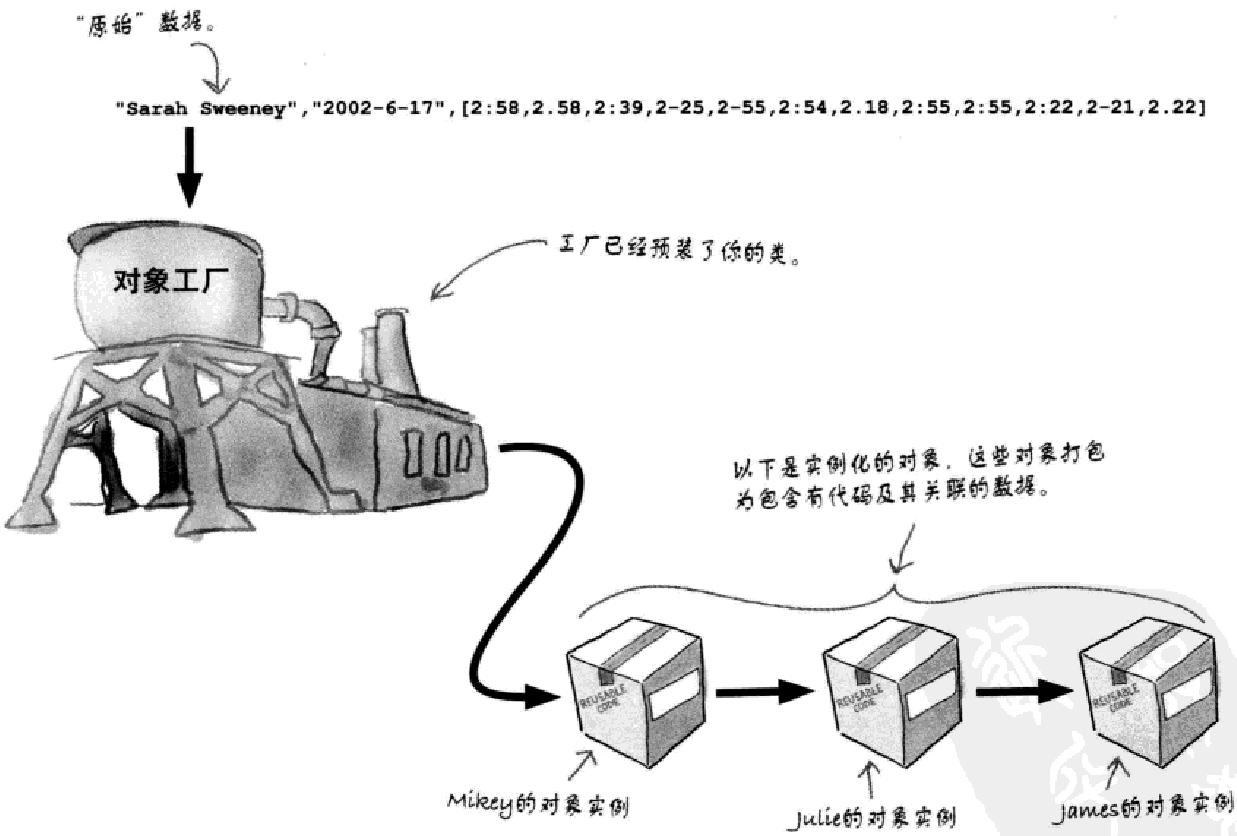
bug更少意味着代码更可维护。

通过使用类，你可以在一个地方维护你的代码和数据，随着代码基的扩大，这确实会带来显著的差别。特别是如果现在正是凌晨4点，而最后期限离你越来越近……

定义一个类

Python遵循标准的面向对象编程模型，提供了一种方法允许将代码及其处理的数据定义为一个类。一旦有了类定义，就可以用它来创建（或实例化）数据对象，它会继承类的特性。

在面向对象世界里，你的代码通常称为类的方法（method），而数据通常称为类的属性（attribute）。实例化的数据对象通常称为实例（instance）。



每个对象都由类创建，并共享一组类似的特性。各个实例中的方法（代码）都相同，但是各个对象的属性（数据）不同，因为属性要由你的原始数据创建。

下面来看Python中如何定义类。

使用class定义类

Python使用**class**创建对象。每个定义的类都有一个特殊的方法，名为`__init__()`，可以通过这个方法控制如何初始化对象。

类中的方法与函数的定义很类似，也就是说，同样使用def来定义。基本形式如下：

类定义从这个关键字开始。

为类指定一个合适的描述性的名字。

```
class Athlete:
    def __init__(self):
        # The code to initialize a "Athlete" object.
            ...
```

不要忘记加冒号！

"init"前面和后面分别有两个下划线。

初始化各个对象的代码放在这里。

创建对象实例

有了类之后，创建对象实例很容易。只需将对类名的调用赋至各个变量。通过这种方式，类（以及`__init__()`方法）提供了一种机制，允许你创建一个定制的工厂函数，用来根据需要创建多个对象实例：

```
a = Athlete()
b = Athlete()
c = Athlete()
d = Athlete()
```

小括号告诉Python要创建一个新的"Athlete"对象，然后赋给一个变量。

所有这些变量都是唯一的，类型都是Athlete。

与C++系列语言不同，Python中没有定义构造函数"new"的概念。Python会为你完成对象构建，然后你可以使用`__init__()`方法定制对象的初始状态。

self的重要性

重申一句：定义一个类时，实际上是在定义一个定制工厂函数，然后可以在你的代码中使用这个工厂函数创建实例：

```
a = Athlete()
```

目标标识符，包含实例的一个引用。

调用类的定制工厂函数。

Python处理这行代码时，它把工厂函数调用转换为以下调用，明确了类、方法（自动设置为`__init__()`）和所处理的对象实例：

```
Athlete().__init__(a)
```

类名

对象实例的目标标识符。

方法名

下面再来看如何在类中定义`__init__()`方法：

```
def __init__(self):
    # The code to initialize an "Athlete" object.
        ...
```

查看Python会如何转换你的对象创建调用。注意到什么了吗？

目标标识符赋至self参数。

这是一个非常重要的参数赋值。如果没有这个赋值，Python解释器无法得出方法调用要应用到哪个对象实例。注意，类代码设计为在所有对象实例间共享：方法是共享的，而属性不共享。`self`参数可以帮助标识要处理哪个对象实例的数据。

PDG

每个方法的第一个参数都是self

实际上，不仅__init__()方法需要self作为它的第一个参数，类中定义的所有其他方法也是如此。

Python要求每个方法的第一个参数为调用对象实例。下面扩展这个示例类，在一个名为thing的对象属性中存储一个值，具体值将在初始化时设置。另外还要扩充一个方法，名为how_big()，它会利用len() BIF返回thing的长度：

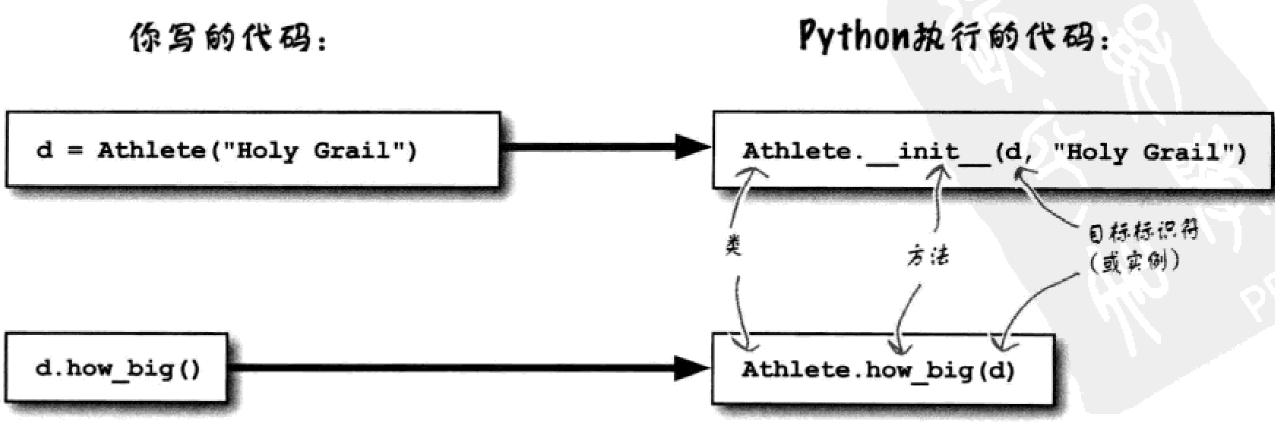
```
class Athlete:
    def __init__(self, value=0):
        self.thing = value
    def how_big(self):
        return(len(self.thing))
```

现在"初始化"代码将所提供的一个值赋至一个名为"self.thing"的类属性。

注意这里使用"self"标识调用对象实例。

"how_big()"方法返回"self.thing"的长度。

在一个对象实例上调用类方法时，Python要求第一个参数是调用对象实例，这往往赋至各方法的self参数。仅凭这一点就足以解释为什么self如此重要，也可以由此说明编写对象方法时为什么所有对象方法的第一个参数必须是self：



An IDLE Session

下面使用IDLE用你定义的新类创建一些对象实例。首先创建一个名为Athlete的很小的类：

```
>>> class Athlete:
        def __init__(self, a_name, a_dob=None, a_times=[]):
                self.name = a_name
                self.dob = a_dob
                self.times = a_times
```

注意这两个参数的缺省值。

初始化这3个属性，使用所提供的参数数据为3个类属性赋值。

定义类之后，创建两个唯一的对象实例，它们会继承Athlete类的特性：

```
>>> sarah = Athlete('Sarah Sweeney', '2002-6-17', ['2:58', '2.58', '1.56'])
>>> james = Athlete('James Jones')
>>> type(sarah)
<class '__main__.Athlete'>
>>> type(james)
<class '__main__.Athlete'>
```

创建两个唯一的选手（"james"使用缺省参数值）。

确认"sarah"和"james"都是选手。

尽管sarah和james都是选手，都由Athlete类的工厂函数创建，但它们存储在不同的内存地址上：

```
>>> sarah
<__main__.Athlete object at 0x14d23f0>
>>> james
<__main__.Athlete object at 0x14cb7d0>
```

这些是计算机上的内存地址，可能与你的计算机上报告的值不同。这里要强调的重点是"sarah"和"james"的内存地址不同。

既然sarah和james都是对象实例，因此可以使用我们熟悉的点记法来访问与各个对象实例关联的属性：

```
>>> sarah.name
'Sarah Sweeney'
>>> james.name
'James Jones'
>>> sarah.dob
'2002-6-17'
>>> james.dob
>>> sarah.times
['2:58', '2.58', '1.56']
>>> james.times
[]
```

"james"对象实例没有"dob"值，所以屏幕上不会显示任何内容。

Sharpen your pencil

以下是你的代码（不过不包括santize()函数，因为这个函数无须修改）。拿出笔来，编写代码来定义Athlete类。除了__init__()方法外，还要定义一个新方法top3()，调用这个方法会返回最快的3个时间。

要调整get_coach_data()函数，返回一个Athlete对象而不是字典，另外不要忘记还要修改print()语句。

在这里写Athlete类代码。

```
def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return({'Name' : templ.pop(0),
                'DOB'  : templ.pop(0),
                'Times': str(sorted(set([sanitize(t) for t in templ]))[0:3])})
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')

print(james['Name'] + "'s fastest times are: " + james['Times'])
```

这里需要做哪些修改来确保这个函数返回一个Athlete对象而不是一个字典？

这行代码也需要修改。

Sharpen your pencil Solution

以下是你的代码（不过不包括santize()函数，因为这个函数无须修改）。拿出笔来，编写代码来定义Athlete类。除了__init__()方法外，还要定义一个新方法top3()，调用这个方法会返回最快的3个时间。要调整get_coach_data()函数，返回一个Athlete对象而不是字典，另外不要忘记还要修改print()语句。

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])
```

这里没有新的内容，因为这个代码直接取自上一个IDLE会话。

没有忘记使用"self"吧？

```
def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return({'Name' : templ.pop(0),
                'DOB' : templ.pop(0),
                'Times': str(sorted(set([sanitize(t) for t in templ]))[0:3])})
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')

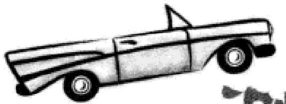
print(james['Name'] + "'s fastest times are: " + james['Times'])
```

（手写修改：return语句中被划掉的字典代码替换为 `Athlete(templ.pop(0), templ.pop(0), templ)`；print语句中被划掉的 `james['Name']` 替换为 `james.name`，`james['Times']` 替换为 `str(james.top3())`。）

删除字典创建代码，替换为Athlete对象创建代码。

使用点记法访问数据。

调用"top3()"方法，将结果显示在屏幕上之前先把结果转换为一个字符串。



测试驱动

对程序做这些修改之后，下面来确保仍然能得到与前面一样的结果。把你的代码加载到IDLE并运行。

coach4.py - /Users/barryp/HeadFirstPython/chapter6/coach4.py

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return(Athlete(templ.pop(0), templ.pop(0), templ))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')
julie = get_coach_data('julie2.txt')
mikey = get_coach_data('mikey2.txt')
sarah = get_coach_data('sarah2.txt')

print(james.name + "'s fastest times are: " + str(james.top3()))
print(julie.name + "'s fastest times are: " + str(julie.top3()))
print(mikey.name + "'s fastest times are: " + str(mikey.top3()))
print(sarah.name + "'s fastest times are: " + str(sarah.top3()))
```

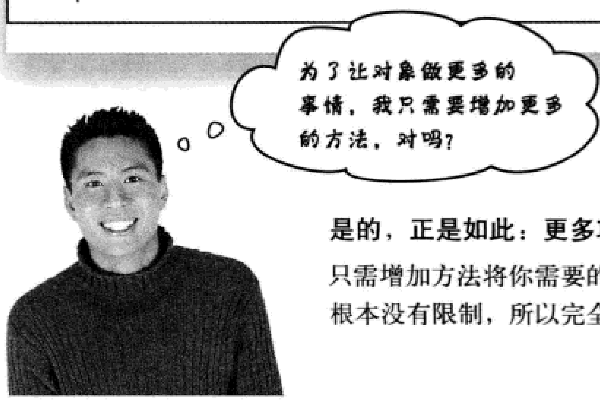
这里没有给出“sanitize()”函数的代码，不过这个函数仍是程序的一部分。

Python Shell

```
>>> ================================ RESTART ================================
>>>
James Lee's fastest times are: ['2.01', '2.16', '2.22']
Julie Jones's fastest times are: ['2.11', '2.23', '2.59']
Mikey McManus's fastest times are: ['2.22', '2.31', '2.38']
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>> |
```

Ln: 30 Col: 4

太棒了！这里没有变化。



是的，正是如此：更多功能 = 更多方法。

只需增加方法将你需要的新功能封装在类中。一个类可以有多少个方法根本没有限制，所以完全由你说了算！

there are no Dumb Questions

问：我不太清楚为什么要把top3()方法编写为返回一个包含3个数据项的列表，而不是一个字符串？如果是一个字符串，肯定更有利于主程序中的print()语句输出，不是吗？

答： 确实是这样，不过那样做就不灵活了。通过返回一个列表（尽管很小），top3()方法允许由调用代码来决定下一步做什么，而不是要求调用者必须处理一个字符串。必须承认，目前这个程序确实要把列表处理为一个字符串，不过并不是所有程序都希望或需要这么做。

问：这个类为什么需要top3()方法？为什么不把最快的3个时间作为一个属性存储在类中，并作为对象创建的一部分来创建这个属性呢？

答： 同样地，最好不要这么做，因为这样做会降低灵活性。如果在对象创建时计算并存储前3个时间，扩展与对象关联的计时列表时就会很困难。

例如，如果在对象创建之后增加更多的计时数据，就需要重新计算前3个时间（因为新的时间可能更快），并更新属性。不过，如果使用top3()方法调用"动态"地计算最快的3个时间，你总能确信所使用的是最新数据。

问：好吧，我明白了。不过，只需要多做一点工作，我就可以在对象创建时进行计算，是吗？

答： 嗯，算是吧……不过我们真的不建议这么做。通过在各个对象的属性中保留原始数据，可以支持类的扩展来满足将来的其他需求（一切都有可能）。如果处理数据并作为对象初始化代码的一部分，说明你已对程序员将如何使用这个类做出了假设，而日后这些假设肯定会对你造成障碍。

问：但是如果除了我以外不会有其他程序员使用我写的定制类呢？

答： 相信我：如果你能把类编写得足够灵活，等你在将来的某个项目中用它来完成某个其他用途时，你会很感激自己的。创建一个类时，你根本无从知道其他程序员会在他们的项目中如何使用这个类。而且，如果你仔细想想，你甚至不会知道你自己将来会如何使用这个类。

问：好吧，我想我已经被你说服了。不过请你告诉我：我该怎么向现有的Athlete对象增加更多时间数据呢？

答： 要做更多事情，只需要增加更多的方法。创建Athlete类之后，扩展这个类来完成更多工作非常容易。只需增加更多的方法。

所以，如果你想向times属性增加一个新的计时值，可以定义一个方法，名为add_time()，让它为你完成这个工作。另外，通过定义一个名为add_times()的方法，还可以增加一个时间列表。这样一来，只需要在代码中做如下调用：

```
sarah.add_time('1.31')
```

就可以向Sarah的计时数据增加一个时间，或者如下调用：

```
james.add_times(['1.21','2.22'])
```

可以向James的数据增加一组时间。

问：不过，既然知道时间是一个列表，我肯定可以写类似下面的代码来完成同样的工作，对吗？

```
sarah.times.append('1.31')
james.times.append(['1.21','2.22'])
```

答： 可以，但是这会带来一场灾难。

问：什么?你为什么会这么说？我的建议并未有任何问题，不是吗？

答： 嗯……这种做法确实可行。不过，编写这样的代码存在一个问题，这就是它会暴露一个事实：计时数据存储在Athlete类的一个列表中。如果你以后改变类的实现，将使用（例如）一个字符串而不是一个列表，就会破坏原先使用这个类的所有代码。另外如果原来有代码利用了计时数据是一个列表这一事实，所有这些代码也将被破坏。

通过用add_time()和add_times()定义你自己的API，可以保持开放性，这样一来，将来还可以改变数据在类中存储的方式（显然，前提是如果确实有必要修改）。需要指出，使用面向对象的原因之一就是要对类的用户隐藏类实现的细节。定义你自己的API正是充分支持了这个设计理念。倘若暴露类实现的内部细节，并且希望程序员利用类的实现细节，这在很大程度上会破坏面向对象的这个基本理念。

Exercise

向你的类增加两个方法。第一个名为add_time()，将一个额外的计时值追加到选手的计时数据。第二个方法add_times()会用一个或多个计时值（提供为一个列表）来扩展一个选手的计时数据。

以下是你目前的Athlete类：请增加代码来实现这两个新方法。

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])
```

在这里增加你的方法。

Sharpen your pencil

先别忙着放下笔！再加几行代码来测试你的新功能：

Exercise Solution

向你的类增加两个方法。第一个名为add_time()，将一个额外的计时值追加到选手的计时数据。第二个方法add_times()会用一个或多个计时值（提供为一个列表）来扩展一个选手的计时数据。

以下是你目前的Athlete类：请增加代码来实现这两个新方法。

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])
```

```
    def add_time(self, time_value):
        self.times.append(time_value)

    def add_times(self, list_of_times):
        self.times.extend(list_of_times)
```

不要忘记使用“self”!!!

将提供的参数追加到现有的计时值列表。

用提供的参数列表扩展现有的计时值列表。

Sharpen your pencil Solution

趁你还拿着笔，再加几行代码来测试你的新功能：

```
vera = Athlete('vera vi')
vera.add_time('1.31')
print(vera.top3())
vera.add_times(['2.22', "1-21", '2:22'])
print(vera.top3())
```

为vera创建一个新的对象实例。

增加一个计时值。

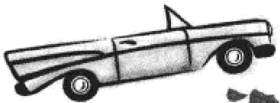
这会显示一个列表，其中只有一个值：1.31。

再增加3个计时值。

现在前3个计时值是：1.21、1.31和2.22。

动手做！

完成这个测试之前，先用更新的Athlete类修改你的代码。



测试驱动

运行现有程序后，在IDLE shell中尝试运行你的测试代码，确认一切正常。

Python Shell

```
>>> ================================ RESTART ================================
>>>
James Lee's fastest times are: ['2.01', '2.16', '2.22']
Julie Jones's fastest times are: ['2.11', '2.23', '2.59']
Mikey McManus's fastest times are: ['2.22', '2.31', '2.38']
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>>
>>> vera = Athlete('Vera Vi')
>>> vera.add_time('1.31')
>>> print(vera.top3())
['1.31']
>>>
>>> vera.add_times(['2.22', "1-21", '2:22'])
>>> print(vera.top3())
['1.21', '1.31', '2.22']
>>> |
```

Ln: 179 Col: 4

不出所料。

创建一个新选手。

增加一个计时值。

显示前3个时间（现在只有一个计时值，所以只能看到一个时间）。

再增加3个计时值。

显示前3个时间（现在才更合理一些）。

太好了，成功了。

你已经把代码与数据打包在一起，并且创建了一个定制类，由这个类可以创建共享相同行为的对象。如果需要额外的功能，还可以增加更多的方法来实现所需的功能。

通过把选手代码和数据封装在一个定制类中，这就创建了一个更可维护的软件。6个月后当你需要修改这个代码时，你会很庆幸自己之前做的这些工作。

干得不错。确实是不小的进步！



是的，Athlete类确实很像列表。

在大多数情况下，你的`Athlete`类确实表现得像是一个列表，另外你增加了一些方法为这个类的用户提供了一些列表功能。你说的没错，在这里确实是重新开始。你的`add_time()`方法是一个“瘦”包装器，它包装了列表的`append()`方法。另外`add_times()`方法实际上是“经过伪装”的列表的`extend()`方法。

事实上，`Athlete`类与Python列表的区别只是`Athlete`类包含`name`和`dob`对象属性。


如果有办法扩展一个内置类来增加定制的属性，那该多好！不过我想这只是异想天开吧……

继承Python内置的list

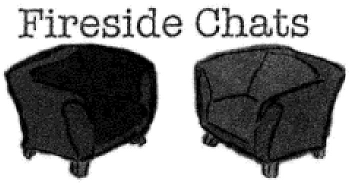
Python的**class**允许你从零开始创建一个定制类，就像创建`Athlete`类那样。不过，**class**还允许通过继承现有的其他类来创建一个类，这也包括用`list`、`set`和`dict`提供的Python内置数据结构类。通过继承创建的这些类称为子类。

有一点非常棒，这就是从一个现有的类（如`list`）继承时，会为你免费提供所有现有的功能。

由于现在这个类实际上只是一个列表，只不过增加了一些属性，也许更好的设计是丢掉你原来的`Athlete`类，把它替换为一个从内置`list`类继承的类，是吗？这种想法当然值得考虑。







今晚话题：继承，也就是说，他看起来很像他的父亲。

定制类：

继承类：

定制类： 程序员都喜欢我，因为他们习惯于在他们的代码中控制一切……你应该了解程序员的：他们就是喜欢写代码。

继承类： 是的，他们确实喜欢编程。不过有时一切都从零开始写起并不是最佳的设计决策。

定制类： 设计！真是好笑！真正的程序员生活中的一切都是代码，不论是吃饭、睡觉、做梦、打鼾都少不了代码，甚至呼吸的都是代码。所有关于设计的言论都是对那些不会编程的人讲的！

继承类： 真的吗？那么，你是说一切都从头开始，重复完成别人已经做过的工作，这种做法在你看来会更好，而这只是因为你相信你的做法就是最好的做法，是吗？你真的这么以为吗？

定制类： 不，不是的，你没有听懂我的意思。这里的重点是控制。如果你一切都从零开始构建，说明一切都在你的控制之中，因为这些都是你的代码。

继承类： 这么说，你很乐意从零开始，即使很久以前已经有人解决过这个问题你也不在乎？

定制类： 当然，特别是要考虑一些定制需求时。在这种情况下，创建一个全新的定制类是唯一的选择。

继承类： 那可不一定，如果你能扩展其他人的类来处理你的定制需求，这也是完全可以的。这样一来，你可以做到两全其美：一方面可以得到继承的功能（这样你就不必从零开始），另外还可以满足定制需求。这是一个双赢的结果。

定制类： 对，没错……对你来说这是双赢，对我可不是。

继承类： 不过这并不是你我的事情，而是为了让程序员的日子更好过，更轻松，即使那些人生来就是要写代码，你认为呢？

定制类： 我想是的，不过我还是定制代码的“忠实粉丝”……

An IDLE Session

下面来看如何继承Python的内置list类。在IDLE的shell中，首先派生内置list类创建一个定制列表，它还有一个名为name的属性：

```
>>> class NamedList(list):
        def __init__(self, a_name):
                list.__init__([])
                self.name = a_name
```

提供一个类名，新类将派生这个类。

初始化所派生的类，然后把参数赋至属性。

定义了NamedList类之后，用它创建一个对象实例，检查对象的类型（使用type() BIF），看看它会提供什么内容（使用dir() BIF）：

```
>>> johnny = NamedList("John Paul Jones")
>>> type(johnny)
<class '__main__.NamedList'>
>>> dir(johnny)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dict__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__module__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'append', 'count', 'extend', 'index', 'insert', 'name', 'pop', 'remove',
'reverse', 'sort']
```

创建一个新的"NamedList"对象实例。

是的，"johnny"是一个"NamedList"。

"johnny"可以做列表能做的所有事情，另外还可以在"name"属性中存储数据。

使用list类提供的一些功能来补充johnny中存储的数据：

```
>>> johnny.append("Bass Player")
>>> johnny.extend(['Composer', "Arranger", "Musician"])
>>> johnny
['Bass Player', 'Composer', 'Arranger', 'Musician']
>>> johnny.name
'John Paul Jones'
```

使用内置list类提供的方法为"NamedList"增加数据。

访问列表数据以及属性数据。

由于johnny是一个列表，所以完全可以对它做列表处理：

```
>>> for attr in johnny:
        print(johnny.name + " is a " + attr + ".")

John Paul Jones is a Bass Player.
John Paul Jones is a Composer.
John Paul Jones is a Arranger.
John Paul Jones is a Musician.
```

"johnny"类似于任何其他列表，所以能够使用列表的地方都可以使用这个对象。

可以确认：John真的很忙。☺

Exercise

下面是“已故”Athlete类的代码。在以下给出的空白处重写这个类，让它继承内置的list类。将这个新类命名为AthleteList。再提供几行代码实际使用这个新类：

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])

    def add_time(self, time_value):
        self.times.append(time_value)

    def add_times(self, list_of_times):
        self.times.extend(list_of_times)
```

在这里写出你的新类代码。

在这里实际使用你的代码。



下面是“已故”Athlete类的代码。在以下给出的空白处重写这个类，让它继承内置的list类。将这个新类命名为AthleteList。再提供几行代码实际使用这个新类：

```
class Athlete:
    def __init__(self, a_name, a_dob=None, a_times=[]):
        self.name = a_name
        self.dob = a_dob
        self.times = a_times

    def top3(self):
        return(sorted(set([sanitize(t) for t in self.times]))[0:3])

    def add_time(self, time_value):
        self.times.append(time_value)

    def add_times(self, list_of_times):
        self.times.extend(list_of_times)
```

不再需要这些方法。

数据本身是计时数据，所以不再需要“times”属性。

类名已经改变。

继承内置的list类。

这里没有新的内容……这段代码与“NamedList”初始化代码很相似。

```
class AthleteList(list):
    def __init__(self, a_name, a_dob=None, a_times=[]):
        list.__init__([])
        self.name = a_name
        self.dob = a_dob
        self.extend(a_times)
    def top3(self):
        return(sorted(set([sanitize(t) for t in self]))[0:3])
```

使用新类的类名。

既然从内置的list继承，所以可以使用list的方法来完成工作。

```
vera = AthleteList('Vera Vi')
vera.append('1.31')
print(vera.top3())
vera.extend(['2.22', "1-21", '2:22'])
print(vera.top3())
```

这个代码可以很好地演练你的新类。

动手做！

在你的代码中，将原来的Athlete类代码替换为新的AthleteList类代码，另外不要忘记修改get_coach_data()来返回一个AthleteList对象实例而不是Athlete对象实例。

there are no Dumb Questions

问：抱歉……不过你不是3分钟前才告诉我不要把类的内部工作暴露给类用户吗？因为你说这从根本上就是一个不好的想法。现在你简直是前后矛盾！怎么回事？

答： 你观察得很仔细。在这种特定的情况下，暴露这个类建立在list之上是完全可以的。这是因为，这个类有意地命名为AthleteList，以此区别更为一般的Athlete类。程序员看到一个类名中包括"list"时，他们通常会认为这个类能表现得像一个列表，另外还可能有一些其他功能。AthleteList正是如此。

问：那么我可以从任何内置的类型继承吗？

答： 可以。

问：那么从多个类继承呢？……Python支持多重继承吗？

答： 是的，不过这方面有些复杂。关于多重继承的详细内容请找一本好的Python参考书看看。

问：我可以从我自己的定制类继承吗？

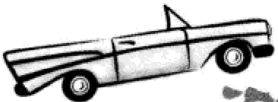
答： 当然可以。这正是关键。你可能会创建一个一般化的类，然后"派生子类"来提供更为特定、更有针对性的功能。

问：可以把我的类放在一个模块文件中吗？

答： 可以，这确实是一个很好的想法，因为这样不仅可以在你自己的多个程序中共享这个类，还可以与其他程序员共享。例如，如果你把AthleteList类保存到一个名为athletelist.py的文件中，可以使用下面这行代码把这个类导入到你的代码中：

```
from athletelist import AthleteList
```

然后就可以正常使用这个类，就好像它是在当前这个程序中定义的一样。另外，如果你创建了一个很有用的类，当然可以把它作为一个单独的模块上传到PyPI与全世界共享。



测试驱动

最后再运行一次这个程序，应该能确认程序现在已经满足要求。在IDLE中执行这个程序来做出确认。

coach6.py - /Users/barryp/HeadFirstPython/chapter6/coach6.py

```python
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

class AthleteList(list):

    def __init__(self, a_name, a_dob=None, a_times=[]):
        list.__init__([])
        self.name = a_name
        self.dob = a_dob
        self.extend(a_times)

    def top3(self):
        return(sorted(set([sanitize(t) for t in self]))[0:3])

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return(AthleteList(templ.pop(0), templ.pop(0), templ))
    except IOError as ioerr:
        print('File error: ' + str(ioerr))
        return(None)

james = get_coach_data('james2.txt')
julie = get_coach_data('julie2.txt')
mikey = get_coach_data('mikey2.txt')
sarah = get_coach_data('sarah2.txt')

print(james.name + "'s fastest times are: " + str(james.top3()))
print(julie.name + "'s fastest times are: " + str(julie.top3()))
print(mikey.name + "'s fastest times are: " + str(mikey.top3()))
print(sarah.name + "'s fastest times are: " + str(sarah.top3()))
```

程序现在会生成教练想要的输出。

Python Shell

```
>>> ================================ RESTART ================================
>>>
James Lee's fastest times are: ['2.01', '2.16', '2.22']
Julie Jones's fastest times are: ['2.11', '2.23', '2.59']
Mikey McManus's fastest times are: ['2.22', '2.31', '2.38']
Sarah Sweeney's fastest times are: ['2.18', '2.21', '2.22']
>>> |
```

Ln: 306 Col: 4

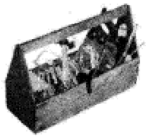
Kelly教练相当满意



通过将类建立在内置的功能之上，你充分利用了Python数据结构的强大功能，同时还提供了具体应用需要的定制解决方案。

对于Kelly教练的数据处理需求，你提供了一个更可维护的解决方案。

好样的！



你的Python工具箱

你已经读完了第6章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- "字典"——这是一个内置的数据结构，允许将数据值与键关联。
- "键"——字典中查找的部分。
- "值"——字典中的数据部分（可以是任何值，也包括另一种数据结构）。

更多Python术语

- "self"——这是一个方法参数，总是指向当前对象实例。

BULLET POINTS

- 使用`dict()`工厂函数或使用`{}`可以创建一个空字典。
- 要访问一个名为`person`的字典中与键`Name`关联的值，可以使用我们熟悉的中括号记法：`person['Name']`
- 类似于列表和集合，Python的字典会随着新数据增加到这个数据结构中而动态扩大。
- 可以先创建一个空字典：`new_d = {}`或`new_d = dict()`然后增加数据`d['Name'] = 'Eric Idle'`来填充字典，或者也可以一次完成以上的全部工作：`new_d = {'Name': 'Eric Idle'}`
- 可以用`class`关键字定义一个类。
- 类方法（代码）与函数的定义基本相同，也就是说，要用`def`关键字定义。
- 类属性（数据）就像是对象实例中的变量。
- 可以在类中定义`__init__()`方法来初始化对象实例。
- 类中定义的每个方法都必须提供`self`作为第一个参数。
- 类中的每个属性前面都必须有`self`，从而将数据与其实例关联。
- 类可以从零开始构建，也可以从Python的内置类或从其他定制类继承。
- 类可以放在一个Python模块中，并上传到PyPI。

7 Web开发

集成在一起



你迟早会希望与很多人分享你的应用。

要做到这一点，你有很多选择。可以把你的代码上传到PyPI，发送大量email邮件，把你的代码放在一个CD或USB上，或者只要有人需要，就直接把应用手动安装在他们的计算机上。听上去要做很多工作……更何况这会让人很头疼。另外，如果你开发出代码的下一个最佳版本又会发生什么情况？你将如何管理代码的更新？面对现实吧：要想出一个有创意的借口确实很困难。幸运的是，你根本不用去找借口，只需创建一个Web应用就行了。另外，正如这一章所要展示的，用Python完成Web开发确实非常轻松。

分享是好事



你要为你的成功付出代价。

就在Kelly教练展示了你的最新程序之后，新的请求潮水般涌来。看起来所有人都想访问教练的数据！

问题在于：要做到这一点，“最好的办法”是什么？

PDG

可以把你的程序放在Web上



你所要的正是“Web应用”。

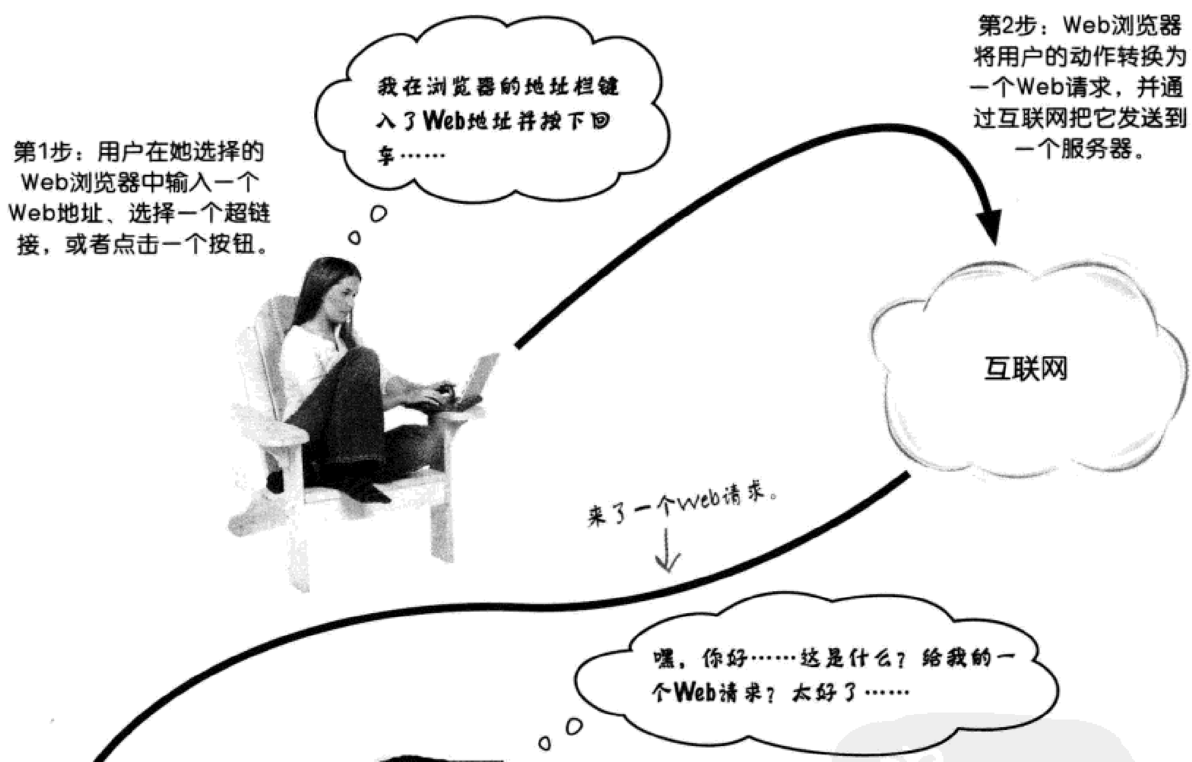
如果将程序开发为一个基于Web的应用，或者简称为Web应用（Webapp），你的程序会有以下特点：

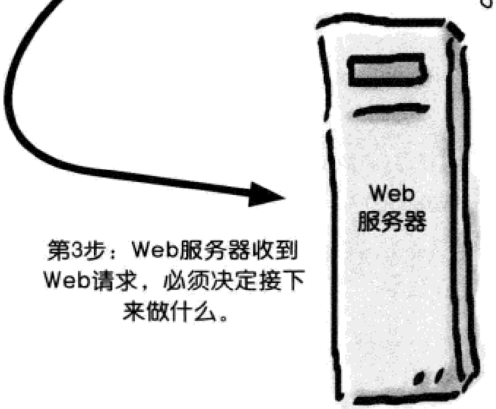
- 能够访问你的网站的每一个人都可以使用这个程序。
- 位于Web服务器上的某个位置。
- 需要新功能时很容易更新。

不过……Web应用究竟是怎样工作的呢？

Web应用特写

不论你在Web上做什么，都离不开请求和响应。Web请求作为某个用户交互的结果由Web浏览器发送到Web服务器。在Web服务器上，会生成Web响应（或应答）并发回到Web浏览器。整个过程可以总结为5个步骤。

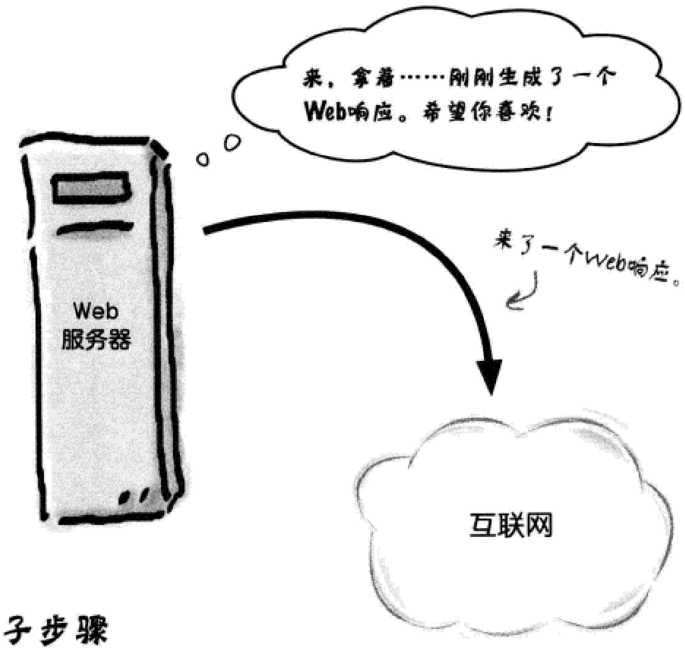




决定接下来做什么

这里可能发生两种情况。如果Web请求所请求的是静态内容（static content），比如一个HTML文件、图像或存储在Web服务器硬盘上的任何其他内容，Web服务器会找到这个资源，并把它作为一个Web响应返回给Web浏览器。

如果请求的是动态内容（dynamic content），也就是说，内容必须动态生成，那么Web服务器会运行一个程序来生成Web响应。



第4步（可能）有很多子步骤

实际上，第4步可能包括多个子步骤，这要看Web服务器如何生成响应。显然，如果服务器所要做的只是找到静态内容并把它发回给浏览器，那么这些子步骤不会太费劲，因为这些都只是文件I/O而已。

不过，如果必须生成动态内容，就要包括以下子步骤：Web服务器要找到所要执行的程序、执行找到的程序，然后捕获程序的输出作为Web响应……再把它发回给还在等待的Web浏览器。

早在Web发展的初期，这个生成动态内容的过程就已经得到标准化，称为通用网关接口（Common Gateway Interface，CGI）。符合这个标准的程序通常称为CGI脚本。



Web应用需要做什么？

下面花些时间来考虑你希望你的Web应用看上去是什么样子，另外在用户的Web浏览器上应该有怎样的表现。然后可以利用这个信息来帮助你确定你的Web应用需要做什么。

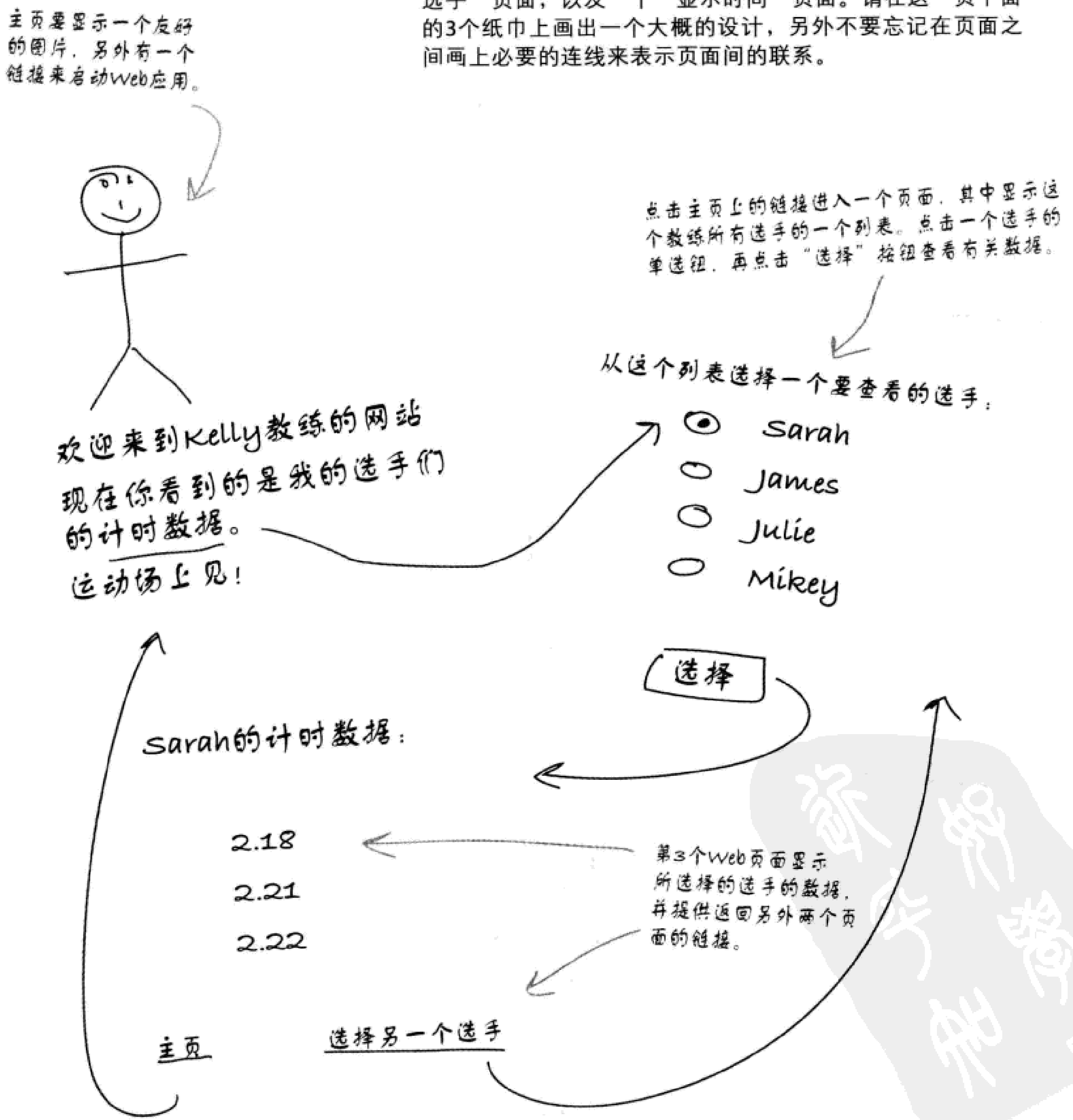


Sharpen your pencil

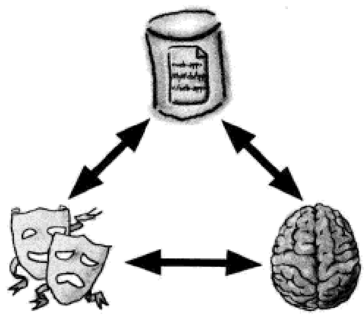
拿出笔来，找几张空白纸巾快速勾画一个简单的Web设计。你可能需要3个Web页面：一个“欢迎”页面、一个“选择选手”页面，以及一个“显示时间”页面。请在这一页下面的3个纸巾上画出一个大概的设计，另外不要忘记在页面之间画上必要的连线来表示页面间的联系。

Sharpen your pencil Solution

拿出笔来，找几张空白纸巾快速勾画一个简单的Web设计。你可能需要3个Web页面：一个“欢迎”页面、一个“选择选手”页面，以及一个“显示时间”页面。请在这一页下面的3个纸巾上画出一个大概的设计，另外不要忘记在页面之间画上必要的连线来表示页面间的联系。



采用MVC设计Web应用



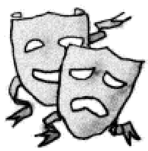
既然已经知道你的Web应用需要提供哪些页面，你肯定会提出下一个问题：建立这些页面的最佳方法是什么？

如果你向10个Web开发人员提出这个问题，会得到10个完全不同的答案，具体答案是什么往往要看你问的是谁。

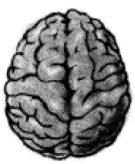
尽管如此，还是会有一个普遍的共识，这就是：好的Web应用应当遵循模型-视图-控制器（Model-View-Controller）模式，这有助于将Web应用的代码分解为易于管理的功能模块（或组件）：



模型
存储（以及有时处理）Web应用数据的代码



视图
格式化和显示Web应用用户界面的代码

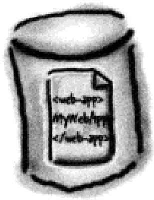


控制器
将Web应用“粘合”在一起并提供业务逻辑的代码

通过遵循MVC模式，可以合理构建你的Web应用，从而当出现新的需求时能够轻松扩展。另外还可以将工作分摊给多个人共同完成，每个人分别负责一个组件。

下面来为你的Web应用构建各个MVC组件。

为数据建模

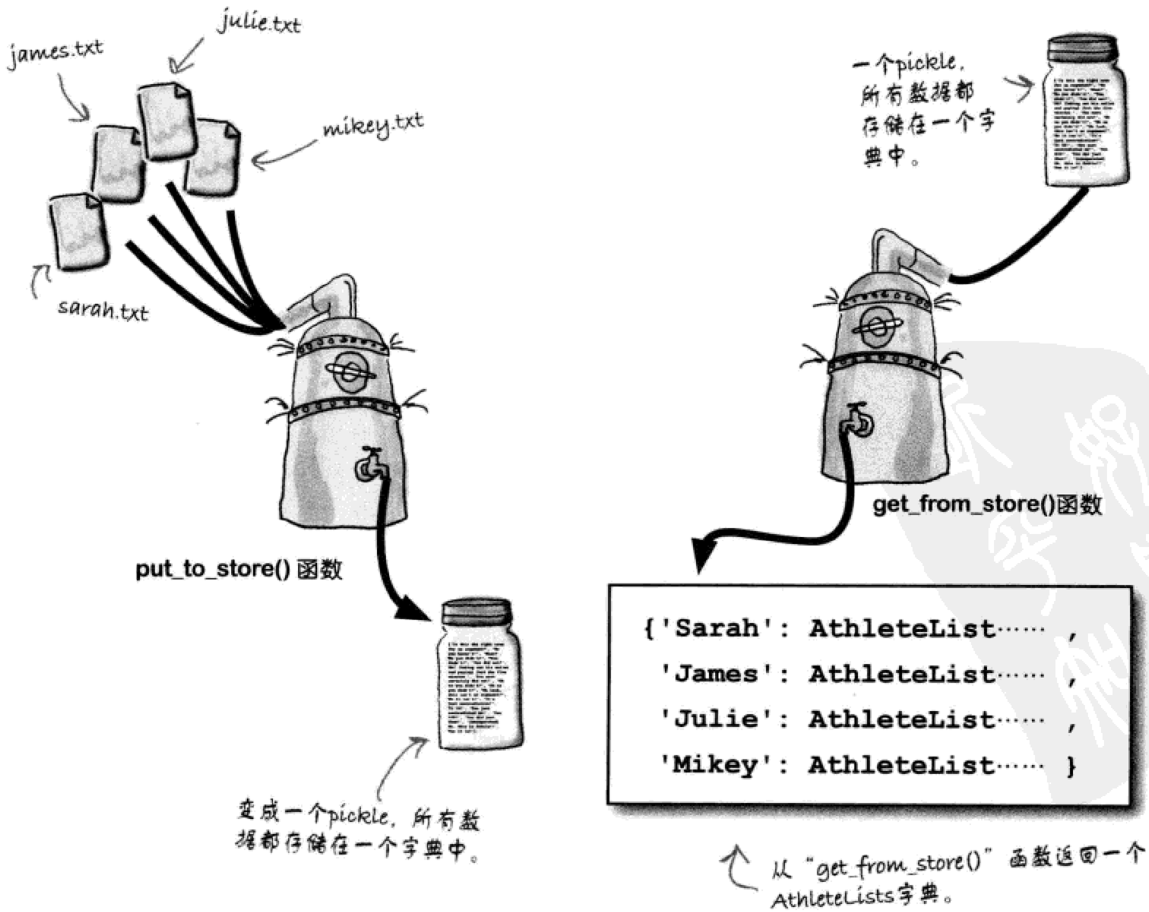


Web服务器需要存储数据的一个副本，在这里数据就是Kelly教练的计时值（来自他的文本文件）。

启动这个Web应用时，需要把文本文件中的数据转换为`AthleteList`对象实例，存储在一个字典中（按选手名索引），然后保存为一个pickle文件。下面把这个功能放在一个名为`put_to_store()`的新函数中。

这个Web应用运行时，pickle文件中的数据可以作为一个字典供应用使用。下面把这个功能放在另一个名为`get_from_store()`的函数中。

启动Web应用： **运行Web应用：**



Exercise

下面是一个新模块（名为athletemodel.py）的大致轮廓，这个模块提供了上一页描述的功能。这里已经提供了部分代码。你的任务是提供put_to_store()和get_from_store()函数的其余代码。不要忘记对所有文件I/O调用加保护。

```
import pickle
from athletelist import AthleteList

def get_coach_data(filename):
    # Not shown here as it has not changed since the last chapter.

def put_to_store(files_list):
    all_athletes = {}
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    return(all_athletes)

def get_from_store():
    all_athletes = {}
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    ...............................................................
    return(all_athletes)
```

调用这个函数时要提供一个文件名列表作为它的唯一参数。

需要在这里编写代码，用文件中的数据填充字典。

不要忘记把这个字典保存到一个pickle中（另外还应当检查文件I/O错误）。

这两个函数都要返回一个AthleteList字典。

从文件得到字典，从而可以返回给调用者。



Exercise Solution

下面是一个新模块（名为athletemodel.py）的大致轮廓，这个模块提供了上一页描述的功能。这里已经提供了部分代码。你的任务是提供put_to_store()和get_from_store()函数的其余代码。不要忘记对所有文件I/O调用加保护。

```
import pickle
from athletelist import AthleteList

def get_coach_data(filename):
    # Not shown here as it has not changed since the last chapter.

def put_to_store(files_list):
    all_athletes = {}
    for each_file in files_list:
        ath = get_coach_data(each_file)
        all_athletes[ath.name] = ath
    try:
        with open('athletes.pickle', 'wb') as athf:
            pickle.dump(all_athletes, athf)
    except IOError as ioerr:
        print('File error (put_and_store): ' + str(ioerr))
    return(all_athletes)

def get_from_store():
    all_athletes = {}
    try:
        with open('athletes.pickle', 'rb') as athf:
            all_athletes = pickle.load(athf)
    except IOError as ioerr:
        print('File error (get_from_store): ' + str(ioerr))
    return(all_athletes)
```

将各个文件转换为一个AthleteList对象实例，并把选手的数据增加到字典。（指向 ath = get_coach_data(each_file)）

每个选手的名字作为字典的“键”，“值”是AthleteList对象实例。（指向 all_athletes[ath.name] = ath）

将整个AthleteList增加到一个pickle中。（指向 pickle.dump(all_athletes, athf)）

不要忘记用一个try/except来保护你的文件I/O代码。

只需将整个pickle读入字典。再没有比这更容易的吧？（指向 all_athletes = pickle.load(athf)）

重申一句……不要忘记使用try/except。

An IDLE Session

下面来测试这个代码，确保它能满足要求。将你的代码键入一个IDLE编辑窗口，并把代码保存到一个文件夹中（这个文件夹还包含有教练的文本文件）。按下F5将代码导入IDLE shell，然后使用`dir()`命令确认导入成功：

```
>>> dir()
['AthleteList', '__builtins__', '__doc__', '__name__', '__package__', 'get_coach_data',
'get_from_store', 'pickle', 'put_to_store']
```

创建一个要处理的文件列表，然后调用`put_to_store()`函数将文件列表中的数据转换为一个字典，存储在一个pickle中：

```
>>> the_files = ['sarah.txt', 'james.txt', 'mikey.txt', 'julie.txt']
>>> data = put_to_store(the_files)
>>> data
{'James Lee': ['2-34', '3:21', '2.34', '2.45', '3.01', '2:01', '2:01', '3:10', '2-22', '2-01',
'2.01', '2:16'], 'Sarah Sweeney': ['2:58', '2.58', '2:39', '2-25', '2-55', '2:54', '2.18',
'2:55', '2:55', '2:22', '2-21', '2.22'], 'Julie Jones': ['2.59', '2.11', '2:11', '2:23',
'3-10', '2-23', '3:10', '3.21', '3-21', '3.01', '3.02', '2:59'], 'Mikey McManus': ['2:22',
'3.01', '3:01', '3.02', '3:02', '3.02', '3:22', '2.49', '2:38', '2:40', '2.22', '2-31']}
```

这里给出所有AthleteList。

现在，你的代码和文本文件所在的同一个文件夹下应该能看到一个`athletes.pickle`文件。应该记得，这个文件是一个二进制文件，所以在IDLE或编辑器中查看这个文件没有太大意义。要访问这个数据，需要使用`put_to_store()`或`get_from_store()`函数返回的字典。

使用数据字典中现有的数据来显示各个选手的名字和出生日期：

```
>>> for each_athlete in data:
        print(data[each_athlete].name + ' ' + data[each_athlete].dob)

James Lee 2002-3-14
Sarah Sweeney 2002-6-17
Julie Jones 2002-8-17
Mikey McManus 2002-2-24
```

通过访问"name"和"dob"属性，可以得到其余的AthleteList数据。

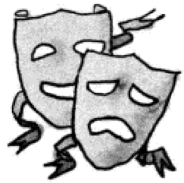
使用`get_from_store()`函数将腌制数据加载到另一个字典中，然后重复前面的代码显示各个选手的名字和出生日期，确认确实能得到期望的结果：

```
>>> data_copy = get_from_store()
>>> for each_athlete in data_copy:
        print(data_copy[each_athlete].name + ' ' + data_copy[each_athlete].dob)

James Lee 2002-3-14
Sarah Sweeney 2002-6-17
Julie Jones 2002-8-17
Mikey McManus 2002-2-24
```

所返回的字典中的数据与期望的一样，与put_to_store()生成的数据完全相同。

查看界面



你已经编写了能够正常工作的模型代码，现在来看视图代码，这会创建Web应用的用户界面（user interface，UI）。

在Web上，用户界面用Web的标记技术HTML来创建。如果你不了解HTML，需要花些时间来熟悉这种非常重要的Web开发技术。网上有很多相关的资料，还有很多非常棒的书可以参考。



[来自市场的声音：这是我们强烈推荐的一本书，可以让你最快地掌握HTML……特别声明，在此没有任何私心和偏见。☺]



YATE：另外一个模板引擎

Head First代码审查小组的朋友们听说你打算编写一些代码来为Web应用的用户界面生成HTML。他们发来一些代码，声称这肯定能让你更轻松地完成工作。这是一个很小的库，名为YATE，其中包括一些生成HTML的辅助函数。这个代码是匆匆写成的，设计初衷是用过后就可以将其“丢掉”，所以审查小组没有对它精心“雕琢”，而是在完成之后就原样交给你了。这个代码有些粗糙，不过应该也能用。

```
from string import Template

def start_response(resp="text/html"):
    return('Content-type: ' + resp + '\n\n')

def include_header(the_title):
    with open('templates/header.html') as headf:
        head_text = headf.read()
    header = Template(head_text)
    return(header.substitute(title=the_title))

def include_footer(the_links):
    with open('templates/footer.html') as footf:
        foot_text = footf.read()
    link_string = ''
    for key in the_links:
        link_string += '<a href="' + the_links[key] + '">' + key + '</a>    '
    footer = Template(foot_text)
    return(footer.substitute(links=link_string))

def start_form(the_url, form_type="POST"):
    return('<form action="' + the_url + '" method="' + form_type + '">')

def end_form(submit_msg="Submit"):
    return('<p></p><input type=submit value="' + submit_msg + '">')

def radio_button(rb_name, rb_value):
    return('<input type="radio" name="' + rb_name +
                             '" value="' + rb_value + '"> ' + rb_value + '<br />')

def u_list(items):
    u_string = '<ul>'
    for item in items:
        u_string += '<li>' + item + '</li>'
    u_string += '</ul>'
    return(u_string)

def header(header_text, header_level=2):
    return('<h' + str(header_level) + '>' + header_text +
           '</h' + str(header_level) + '>')

def para(para_text):
    return('<p>' + para_text + '</p>')
```

这里没有多少帮助，只有代码。没有注释、没有解释、没有文档，什么都没有！

Long Exercise

在继续学习这一章其余的内容之前，先来了解这个yate代码。对于这里给出的每一个代码块，请在给出的空白处提供一个描述，说明你认为这一部分要做些什么。

先花点时间在Python文档中查找"Template"模块的有关内容。

在空白处写出你的解释。

```
from string import Template
```

....

....

```
def start_response(resp="text/html"):
    return('Content-type: ' + resp + '\n\n')
```

已经为你填上了一个解释。

这个函数需要一个（可选的）字符串作为参数，用它来创建一个CGI "Content-type:"行，参数缺省值是"text/html"。

```
def include_header(the_title):
    with open('templates/header.html') as headf:
        head_text = headf.read()
    header = Template(head_text)
    return(header.substitute(title=the_title))
```

....

....

....

```
def include_footer(the_links):
    with open('templates/footer.html') as footf:
        foot_text = footf.read()
    link_string = ''
    for key in the_links:
        link_string += '<a href="' + the_links[key] + '">' + key +
                        '</a>    '
    footer = Template(foot_text)
    return(footer.substitute(links=link_string))
```

....

....

....

PDG

```
def start_form(the_url, form_type="POST"):
    return('<form action="' + the_url + '" method="' + form_type + '">')
```

....

....

```
def end_form(submit_msg="Submit"):
    return('<p></p><input type=submit value="' + submit_msg + '"></form>')
```

....

....

```
def radio_button(rb_name, rb_value):
    return('<input type="radio" name="' + rb_name +
                  '" value="' + rb_value + '"> ' + rb_value + '<br />')
```

....

....

```
def u_list(items):
    u_string = '<ul>'
    for item in items:
        u_string += '<li>' + item + '</li>'
    u_string += '</ul>'
    return(u_string)
```

....

....

....

```
def header(header_text, header_level=2):
    return('<h' + str(header_level) + '>' + header_text +
           '</h' + str(header_level) + '>')
```

....

....

```
def para(para_text):
    return('<p>' + para_text + '</p>')
```

....

....

Long Exercise Solution

在继续学习这一章其余的内容之前，先来了解这个yate代码。对于这里给出的每一个代码块，请在给出的空白处提供一个描述，说明你认为这一部分要做些什么。

```
from string import Template
```

从标准库的“string”模块导入“Template”类。它支持简单的字符串替换模板。

注意“resp”的缺省值。

```
def start_response(resp="text/html"):
    return('Content-type: ' + resp + '\n\n')
```

这个函数需要一个（可选的）字符串作为参数，用它来创建一个CGI “Content-type:”行，参数缺省值是“text/html”。

打开模板文件（HTML），读入文件，换入所提供的“标题”。

```
def include_header(the_title):
    with open('templates/header.html') as headf:
        head_text = headf.read()
    header = Template(head_text)
    return(header.substitute(title=the_title))
```

这个函数需要一个字符串作为参数，用在HTML页面最前面的标题中。页面本身存储在一个单独的文件“templates/header.html”中，可以根据需要替换标题。

打开模板文件（HTML），读入文件，换入“the links”中提供的HTML链接字典。

将链接字典转换为一个字符串，然后再换入模板。

这看起来有点怪异，不过这是HTML在字符串中加入空格的一种强制做法。

```
def include_footer(the_links):
    with open('templates/footer.html') as footf:
        foot_text = footf.read()
    link_string = ''
    for key in the_links:
        link_string += '<a href="' + the_links[key] + '">' + key +
                       '</a>    '
    footer = Template(foot_text)
    return(footer.substitute(links=link_string))
```

与“include_header”函数类似，这个函数使用一个字符串作为参数，来创建一个HTML页面的尾部。页面本身存储在一个单独的文件“templates/footer.html”中，参数用于动态地创建一组HMTL链接标记。从这些标记的使用来看，参数应当是一个字典。

这一般是"POST"或"GET"。

```
def start_form(the_url, form_type="POST"):
    return('<form action="' + the_url + '" method="' + form_type + '">')
```

这个函数返回表单最前面的HTML，允许调用者指定URL（表单数据将发送到这个URL），还可以指定所要使用的方法。

```
def end_form(submit_msg="Submit"):
    return('<p></p><input type=submit value="' + submit_msg + '"></form>')
```

这个函数返回表单末尾的HTML标记，同时还允许调用者定制表单"submit"（提交）按钮的文本。

```
def radio_button(rb_name, rb_value):
    return('<input type="radio" name="' + rb_name +
                     '" value="' + rb_value + '"> ' + rb_value + '<br />')
```

给定一个单选钮名和值，创建一个HTML单选钮（通常包括在一个HTML表单中）。注意：两个参数都是必要的。

一个简单的"for"循环就可以达到目的。

```
def u_list(items):
    u_string = '<ul>'
    for item in items:
        u_string += '<li>' + item + '</li>'
    u_string += '</ul>'
    return(u_string)
```

给定一个项列表，这个函数会把该列表转换为一个HTML无序列表。一个简单的"for"循环就可以完成全部工作，每次迭代会向UL元素增加一个LI元素。

```
def header(header_text, header_level=2):
    return('<h' + str(header_level) + '>' + header_text +
           '</h' + str(header_level) + '>')
```

创建并返回一个HTML标题标记（H1、H2、H3等），默认为2级标题。"header_text"参数是必要的。

```
def para(para_text):
    return('<p>' + para_text + '</p>')
```

用HTML段落标记包围一个文本段（一个字符串）。好像有些没必要，是不是？

there are no Dumb Questions

问：include_header()和include_footer()函数中用到的HTML模板放在哪里？

答：这些HTML模板包含在yate模板的下载包里。可以从Head First Python支持网站下载，把它们放在你选择的一个文件夹中。

问：为什么要用yate？为什么不直接把我需要的HTML放在代码里，根据需要利用print()来生成呢？

答：当然也可以那么做，不过那种做法不如这里给出的方法灵活。而且（从我们之前的痛苦经历来看），使用一大堆print()语句来生成HTML确实是可以的，但是这会让你的代码变得像是一团乱麻。

问：你这么做只是因为你在使用MVC，是吗？

答：部分来讲是这样。之所以遵循MVC模式，原因是这样可以确保模型代码与视图代码分离，而且它们都能与控制器代码分离。不论项目的规模有多大，只要遵循MVC，你就会倍感轻松。

问：不过，对于这么小的一个项目使用MVC确实有些大材小用吧？

答：我们可不这么认为，因为你的Web应用肯定还会扩展，等你需要增加更多特性时，MVC的“职责分离”就会成为一大亮点。

An IDLE Session

下面来进一步了解yate模块。下载代码并放入一个易于查找的文件夹后，将模块加载到IDLE中，按下F5来执行。下面首先测试start_response()函数。CGI标准指出，每一个Web响应都必须有一个首部行来指出请求中包含的数据类型，这可以用start_response()控制：

```
>>> start_response()
'Content-type: text/html\n\n'
>>> start_response("text/plain")
'Content-type: text/plain\n\n'
>>> start_response("application/json")
'Content-type: application/json\n\n'
```

默认的CGI响应首部，下面是另外几种类型。

include_header()函数生成一个Web页面的开始部分，允许定制页面的标题：

```
>>> include_header("Welcome to my home on the web!")
'<html>\n<head>\n<title>Welcome to my home on the web!</title>\n<link type="text/css"
rel="stylesheet" href="/coach.css" />\n</head>\n<body>\n<h1>Welcome to my home on the web!</
h1>\n'
```

这看起来有点乱，不过不用担心。这会由你的Web浏览器来处理，而不是你。Web浏览器可以毫不费力地处理这个HTML。注意这里包含了一个CSS文件的链接（有关的更多内容将在后面介绍）。

`include_footer()`函数会生成一个Web页面末尾的HTML，并提供链接（如果已经提供一个链接字典）。倘若字典为空，就不会包含链接HTML：

```
>>> include_footer({'Home': '/index.html', 'Select': '/cgi-bin/select.py'})
'<p>\n<a href="/index.html">Home</a>    <a href="/cgi-bin/select.
py">Select</a>    \n</p>\n</body>\n</html>\n'
>>> include_footer({})
'<p>\n\n</p>\n</body>\n</html>\n'
```

这里包含了链接，下面则没有包含链接。

`start_form()`和`end_form()`函数会建立一个HTML表单，并用参数（如果提供有参数）来调整所生成的HTML的内容：

```
>>> start_form("/cgi-bin/process-athlete.py")
'<form action="/cgi-bin/process-athlete.py" method="POST">'
>>> end_form()
'<p></p><input type=submit value="Submit"></form>'
>>> end_form("Click to Confirm Your Order")
'<p></p><input type=submit value="Click to Confirm Your Order"></form>'
```

这个参数允许指定服务器上的程序名，表单数据将发送到这个程序。

HTML单选钮很容易创建，可以用`radio_button()`函数来创建：

```
>>> for fab in ['John', 'Paul', 'George', 'Ringo']:
        radio_button(fab, fab)

'<input type="radio" name="John" value="John"> John<br />'
'<input type="radio" name="Paul" value="Paul"> Paul<br />'
'<input type="radio" name="George" value="George"> George<br />'
'<input type="radio" name="Ringo" value="Ringo"> Ringo<br />'
```

你最喜欢哪一个？请从单选钮列表中选择。

用`u_list()`函数可以轻松地创建无序列表：

```
u_list(['Life of Brian', 'Holy Grail'])
'<ul><li>Life of Brian</li><li>Holy Grail</li></ul>'
```

同样的，在你看来可能有些费解，不过对你的Web浏览器来说则是小菜一碟。

`header()`函数允许快速建立选定级别的HTML标题（默认级别为2）：

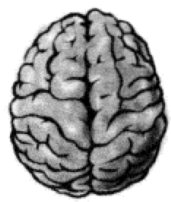
```
>>> header("Welcome to my home on the web")
'<h2>Welcome to my home on the web</h2>'
>>> header("This is a sub-sub-sub-sub heading", 5)
'<h5>This is a sub-sub-sub-sub heading</h5>'
```

这里没有太多惊喜，不过确实能用。下一个函数也是这样。

最后，但不是不重要，`para()`函数会把一个文本块包围在HTML段落标记中间：

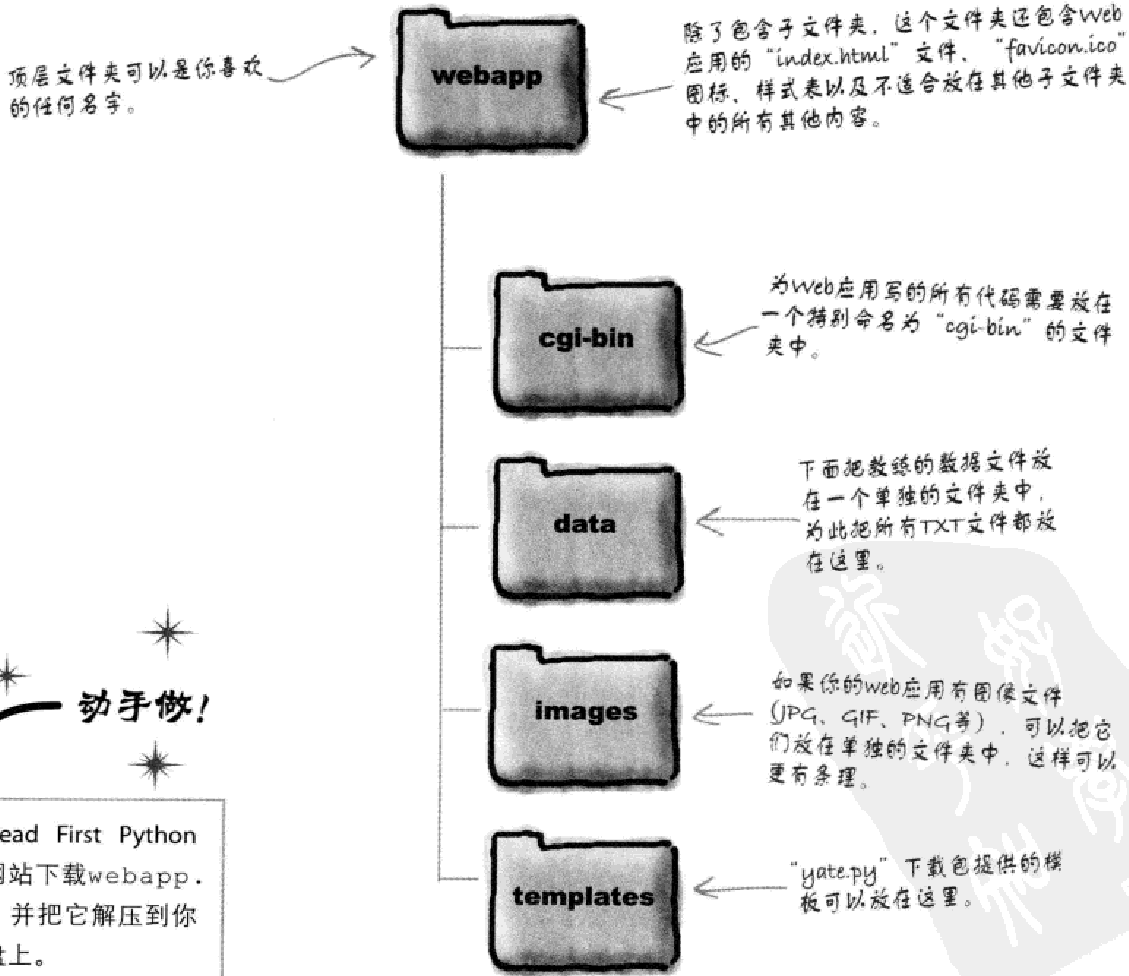
```
>>> para("Was it worth the wait? We hope it was...")
'<p>Was it worth the wait? We hope it was...</p>'
```

控制你的代码



模型代码已经准备好，而且你已经清楚地了解如何利用yate模块来帮助建立视图代码。现在该用一些控制器代码把它们“黏合”在一起了。

首要的事情是：你需要合理安排Web应用的目录结构，保证它有条理有组织。说实在的，在这方面做出的任何努力（尽管只需多加一点考虑），都会在日后大大增强你扩展Web应用的能力。下面是Head First Labs推荐的一个文件夹结构。



动手做！

请到Head First Python支持网站下载webapp.zip，并把它解压到你的硬盘上。

CGI让Web服务器运行程序

通用网关接口（Common Gateway Interface，CGI）是一个Internet标准，允许Web服务器运行一个服务器端程序，称为CGI脚本。

一般地，CGI脚本都放在一个名为cgi-bin的特殊文件夹下，这样Web服务器才能知道在哪里查找CGI脚本。有些操作系统中（主要是类UNIX的系统），CGI脚本必须先设置为可以执行，Web服务器才能执行这些CGI脚本对Web请求做出响应。

稍后还会介绍更多有关内容。





所有Web应用都要在Web服务器上运行。

实际上这个世界的所有Web服务器都支持CGI。不论你在运行Apache、IIS、nginx、Lighttpd还是其他的服务器，它们都支持运行用Python编写的CGI脚本。

不过如果这里使用这些工具就有些大材小用了。教练不可能同意那么费劲地下载、解压、安装、配置和管理这些庞大的工具。

好在，Python提供了它自己的Web服务器，这个Web服务器包含在`http.server`库模块中。请查看`webapp.zip`下载包的内容：它提供了一个支持CGI的Web服务器，名为`simplehttpd.py`。

用Python构建一个Web服务器必须有这5行代码。

导入HTTP服务器和CGI模块。

指定一个端口。

创建一个HTTP服务器。

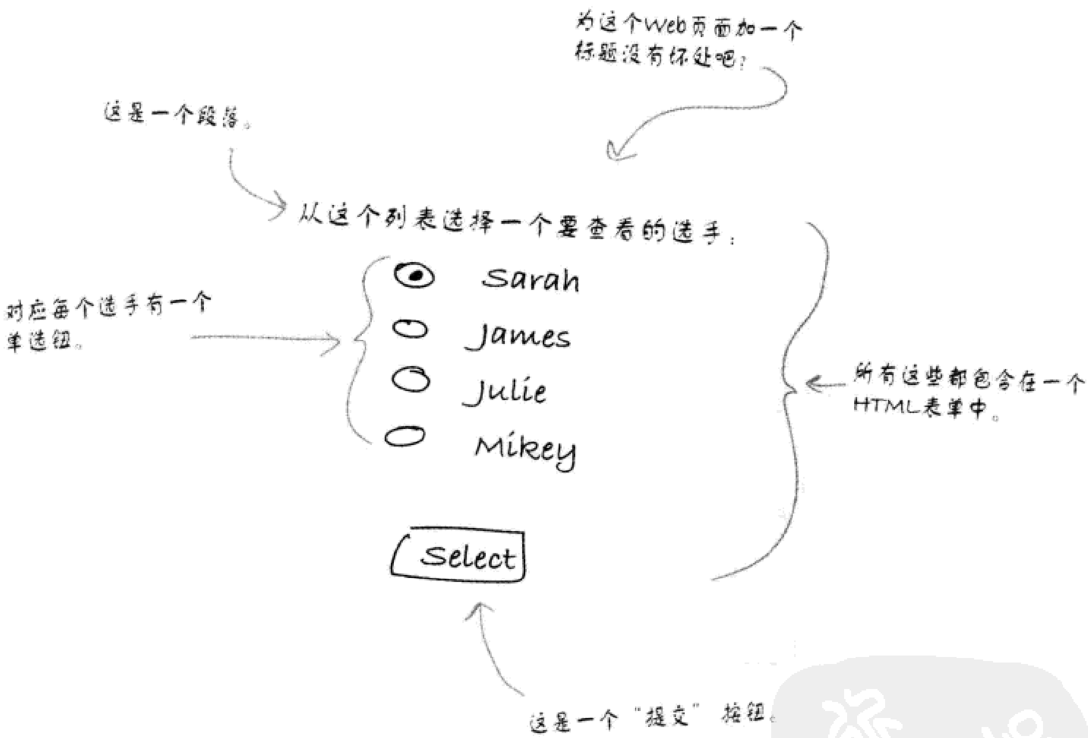
显示一个友好的消息，并启动服务器。

```
from http.server import HTTPServer, CGIHTTPRequestHandler

port = 8080
httpd = HTTPServer(('', port), CGIHTTPRequestHandler)
print("Starting simple_httpd on port: " + str(httpd.server_port))
httpd.serve_forever()
```

显示选手列表

下面创建一个名为`generate _ list.py`的程序，Web服务器执行这个程序时，会动态生成一个类似下面的HTML Web页面：



用户点击对应一个选手的单选钮并点击"Select"(选择)按钮来选择这个选手时，会把一个新的Web请求发送到Web服务器。这个新的Web请求包含有按下了哪个单选钮的有关数据，还包含一个CGI脚本的名（表单数据将发送到这个CGI脚本）。

应该记得，所有CGI脚本都要放在Web服务器上的`cgi-bin`文件夹中。考虑到这一点，要确保你的`generate_list.py` CGI脚本将其数据发送到另一个名为`cgi-bin/generate_timing_data.py`的程序。

池塘谜题



你的任务就是从池塘里取出代码，把它们放在CGI脚本中的空白处。这里的所有代码不能多次使用。你的目标是建立一个CGI脚本，从而生成与上一页手绘设计一致的HTML页面。

我已经为你开了头。

导入你需要的模块。你已经见过"athletemodel"和"yate"。利用"glob"模块可以向操作系统查询一个文件名列表。

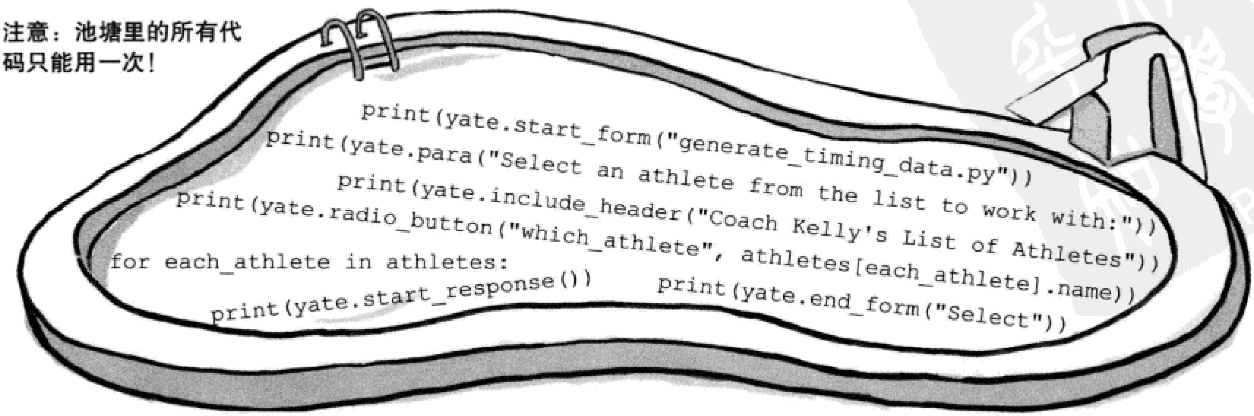
```
import athletemodel
import yate
import glob

data_files = glob.glob("data/*.txt")
athletes = athletemodel.put_to_store(data_files)
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
print(yate.include_footer({"Home": "/index.html"}))
```

使用"put_to_store()"函数由数据文件列表创建一个选手字典。

在生成的HTML页面最下面增加一个链接，指向主页。

注意：池塘里的所有代码只能用一次！



池塘谜题答案



你的任务就是从池塘里取出代码，把它们放在CGI脚本中的空白处。这里的所有代码不能多次使用。你的目标是建立一个CGI脚本，从而生成与上一页手绘设计一致的HTML页面。

```
import athletemodel
import yate
import glob

data_files = glob.glob("data/*.txt")
athletes = athletemodel.put_to_store(data_files)

print(yate.start_response())
print(yate.include_header("Coach Kelly's List of Athletes"))
print(yate.start_form("generate_timing_data.py"))
print(yate.para("Select an athlete from the list to work with:"))

for each_athlete in athletes:
    print(yate.radio_button("which_athlete", athletes[each_athlete].name))
print(yate.end_form("Select"))

print(yate.include_footer({"Home": "/index.html"}))
```

总是从一个Content-type行开始。

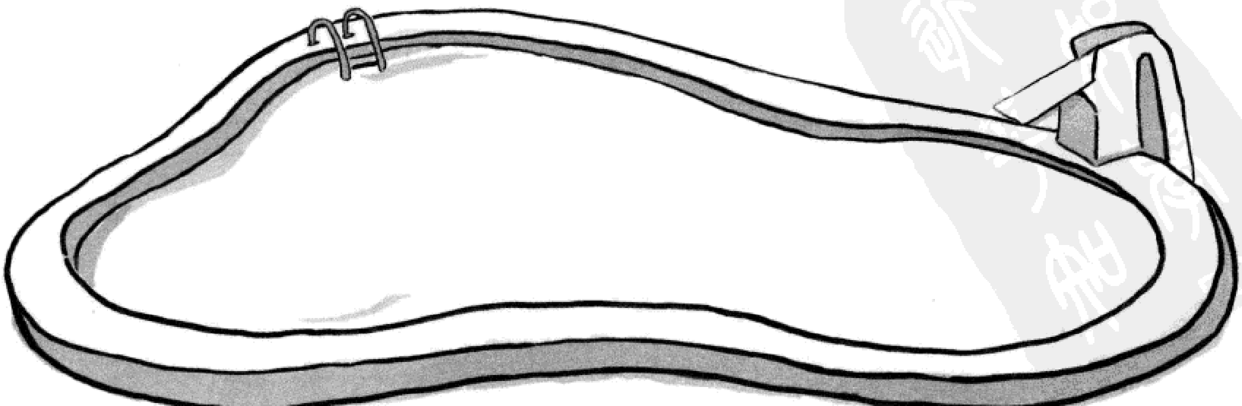
开始生成Web页面，提供一个合适的标题。

开始生成表单，提供要链接的服务器端程序的名。

这是一个段落，告诉你的用户要做什么。

为各个选手分别生成一个单选钮。

生成表单的最后创建一个定制的"提交"按钮。





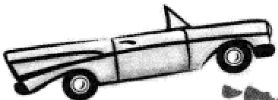
下一步做什么取决于你的Web服务器在哪个操作系统上运行。

如果在Windows上运行，那么不用继续读下面的内容，你可以直接进入测试驱动。不过，如果在运行一个基于Unix系统（如Linux、Mac OS X或BSD），需要做两件事来准备执行你的CGI脚本：

1. 使用chmod +x命令设置CGI的可执行权限位。
2. 在程序最前面增加以下面这行代码：

```
#!  /usr/local/bin/python3
```

在你的终端窗口键入chmod +x generate_list.py来设置可执行权限位。这个工作只需做一次。



测试驱动

为了测试你的CGI脚本，需要启动一个Web服务器并运行。`simplehttpd.py`的代码包含在`webapp.zip`下载包中。解开这个ZIP文件后，从`webapp`文件夹打开一个终端窗口，启动你的Web服务器：

File Edit Window Help WebServerOnWindows

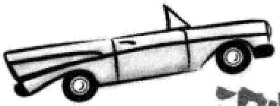
```
c:\Python31\python.exe simplehttpd.py
Starting simple_httpd on port: 8080
```

基于Windows的系统上使用这个命令。

File Edit Window Help WebServerOnUNIX

```
$ python3 simple_httpd.py
Starting simple_httpd on port: 8080
```

基于Unix的系统上使用这个命令。



测试驱动（续）

Web服务器运行后，下面加载Kelly教练的主页来运行这个Web应用。你已经在计算机的端口8080运行了Web服务器，所以需要在Web浏览器中使用以下Web地址：`http://localhost:8080`。



教练的主页出现在浏览器中。这个页面名为“index.html”，包含在“webapp.zip”下载包中。

……你的Web服务器开始工作，它会把它处理的所有Web请求记录下来（显示在屏幕上）。

“timing data”（计时数据）超链接在等着你点击。



当然，点击主页上的链接会运行Web服务器上的generate_list.py程序，这会将Kelly教练的选手显示为一个单选钮列表。



可以点击Home（主页）超链接返回到教练的主页，或者从列表中选择一个选手（点击相应的单选钮），然后按下“Select”（选择）按钮继续。

选择一个选手，并按下Select（选择）按钮。会发生什么？

可怕的404错误！

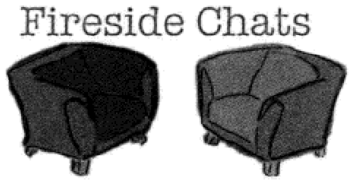
唉呀！你的Web服务器响应了一个“404”错误码，这就是在告诉你请求出了问题。Web服务器实际上在通知你它无法找到Web浏览器请求的资源，所以向你指出犯了一个错误：



查看Web服务器的控制台窗口，确认是由于将表单数据提交到`generate_timing_data.py`导致了失败。

这并不奇怪，因为你还没有写那个代码！所以……问题并没有最初看起来那么糟糕。在这种情况下确实应该显示“404”错误，所以你的`generate_list.py` CGI并没有问题。你需要的只是另一个CGI脚本的代码。

如果创建了所需的CGI脚本，一切又会回归正轨。



今晚话题：做不做CGI，这是个问题。

Python程序：

听着：你实际上和我没有太大不同，只不过你在Web服务器上工作，而我可以在任何地方工作。

特殊？你只能在Web上工作，别处都不行。这怎么能叫“特殊”呢？

真是无稽之谈！事实上，你只能在Web上工作，在别处使用时很快就会崩溃。你甚至无法控制你自己的I/O。

比方说 [窃笑]生成HTML形式的文本？这太费劲了……

噢，别自以为是了！你只是一个普通的程序，就像我一样。我也可以生成HTML，只是我不想而已。

我想是的……

嗯……应该是吧。

Python CGI脚本：

是的。我喜欢把自己想成特殊人物。

因为如今所有奇妙的事情都发生了Web上，而我就是专门为Web设计、优化、调整和实现的。因为Web很棒，所以很自然的，我当然也很棒。知道了吧，这就是我特殊的原因。

我不需要控制我的输入和输出。我有一个友好的Web服务器来为我照管这些事情。我的输入来自Web服务器，我的输出也会送到Web服务器。这种安排使我可以把注意力放在重要的事情上。

随你怎么想。正是HTML让整个互联网得以运转，而我非常擅长根据需要动态地生成HTML。如果没有我，Web将是一个毫无生机的地方。

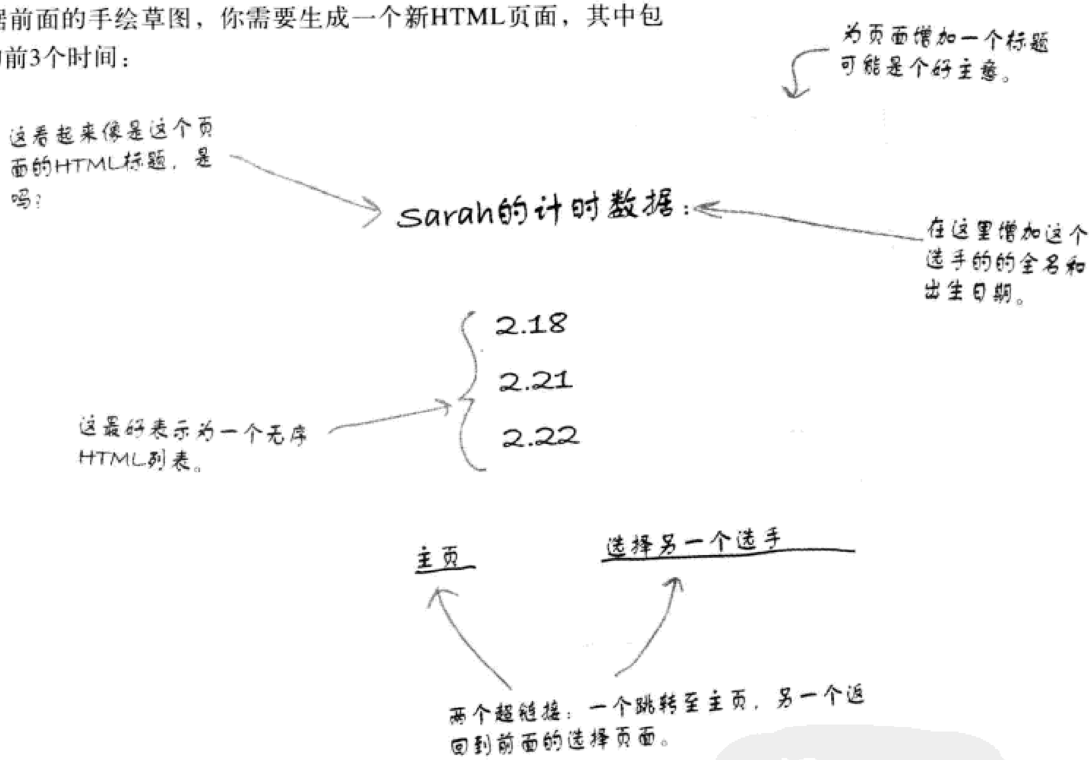
如果你真的生成了HTML，你会把它显示在哪里……比如说浏览器中？

为此你需要依靠一个友好的Web服务器来帮忙，对不对？

那就会让你成为一个CGI脚本。所以你也一样特殊，证明完毕。

创建另一个CGI脚本

下面花点时间来回忆一下`generate_timing_data.py` CGI脚本需要做些什么。根据前面的手绘草图，你需要生成一个新HTML页面，其中包含所选选手的前3个时间：



但是你怎么知道选择了哪个选手呢？

点击一个单选钮然后按"选择"按钮时，会有一个新的Web请求发送到服务器。这个Web请求会指定要执行的CGI脚本（在这里，这个脚本就是`generate_timing_data.py`），还会提供表单的数据。Web服务器会把表单数据作为输入发送到这个CGI脚本。在你的代码中，可以使用Python的CGI模块来访问表单数据，这也是标准库中的一个模块：

导入"cgi"库。

```
import cgi
form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value
```

获取所有表单数据，并放在一个字典中。

从表单数据访问一个指定的数据。

Sharpen your pencil

现在来实际操练你刚掌握的Web开发技能。拿出笔来，写出generate_timing_data.py CGI脚本的代码。这与generate_list.py的代码没有太大不同，所以现有的很多代码都应该能重用。

在这里写出新CGI脚本的代码。

Sharpen your pencil Solution

现在来实际操练你刚掌握的Web开发技能。拿出笔来，写出`generate_timing_data.py` CGI脚本的代码。这与`generate_list.py`的代码没有太大不同，所以现有的很多代码都应该能重用。

```
#! /usr/local/bin/python3

import cgi

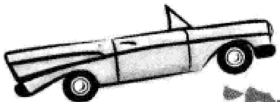
import athletemodel
import yate

athletes = athletemodel.get_from_store()

form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value

print(yate.start_response())
print(yate.include_header("Coach Kelly's Timing Data"))
print(yate.header("Athlete: " + athlete_name + ", DOB: " +
                  athletes[athlete_name].dob + "."))
print(yate.para("The top times for this athlete are:"))
print(yate.u_list(athletes[athlete_name].top3()))
print(yate.include_footer({"Home" : "/index.html",
                           "Select another athlete": "generate_list.py"}))
```

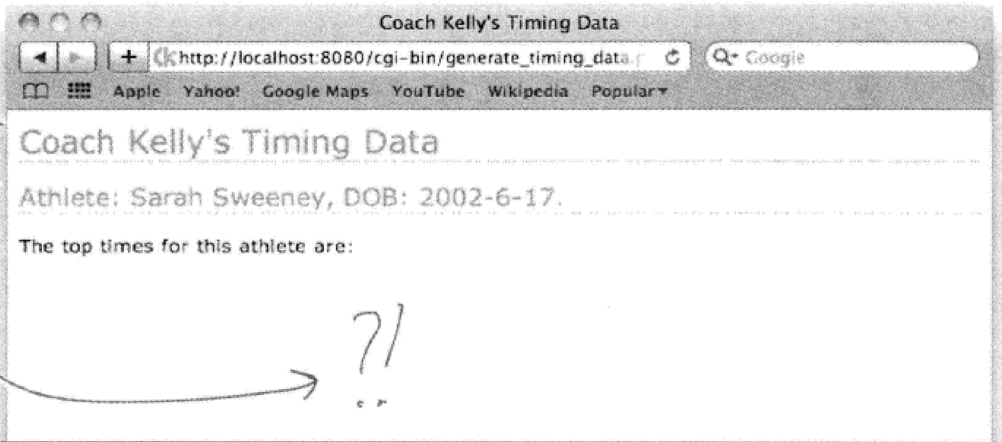
- 只有基于Unix的系统才需要这行代码。
- 导入你要使用的库和模块。
- 从模型得到数据。
- 你在处理哪个选手的数据？
- 这里没有新内容。
- 获取选手的名字和出生日期。
- 将前3个时间转换为一个无序HTML列表。
- 这个web页面的最下面有两个链接。
- 这个链接返回前一个CGI脚本。



测试驱动

注意：如果你使用的是基于Unix的系统，不要忘记增加“chmod +x generate_timing_data.py”设置可执行权限位。

你的Web服务器应该还在运行。如果没有运行，请再次启动Web服务器。在你的Web浏览器中，返回到教练的主页，然后选择显示选手列表的超链接，选择Sarah，再按下“选择”按钮。



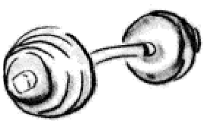
这些看起来都是正常的。

哈，怎么回事？这里少了些东西。Sarah的前3个时间到哪里去了？

```
File Edit Window Help HoustonWeHaveAProblem
$ python3 simple_httpd.py
Starting simple_httpd on port: 8080
localhost - - [12/Sep/2010 14:30:03] "GET / HTTP/1.1" 200 -
localhost - - [12/Sep/2010 14:30:03] "GET /coach.css HTTP/1.1" 200 -
localhost - - [12/Sep/2010 14:30:03] "GET /images/coach-head.jpg HTTP/1.1" 200 -
localhost - - [12/Sep/2010 14:30:03] "GET /favicon.ico HTTP/1.1" 200 -
localhost - - [12/Sep/2010 14:45:16] "GET /cgi-bin/generate_list.py HTTP/1.1" 200 -
localhost - - [12/Sep/2010 16:12:27] "GET /cgi-bin/generate_list.py HTTP/1.1" 200 -
localhost - - [12/Sep/2010 16:12:29] "POST /cgi-bin/generate_timing_data.py HTTP/1.1" 200 -
Traceback (most recent call last):
  File "/Users/barryp/HeadFirstPython/chapter7/cgi-bin/generate_timing_data.py", line 21, in <module>
    print(yate.u_list(athletes[athlete_name].top3()))
TypeError: 'list' object is not callable
localhost - - [12/Sep/2010 16:12:29] CGI script exit status 0x100
```

Web服务器的日志信息能告诉你什么吗？

你的CGI遇到一个TypeError异常，不过除非查看Web服务器的日志屏幕，仅从Web浏览器屏幕我们看不出哪里出了问题。



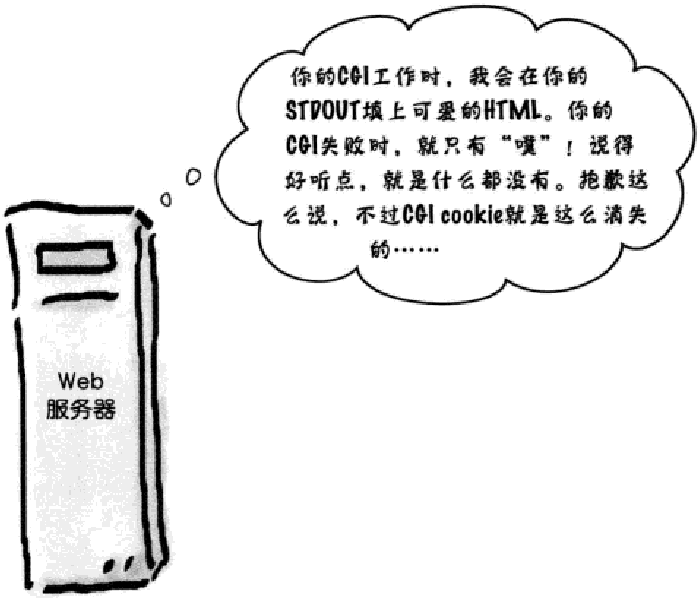
BRAIN BARBELL

你认为这里的问题是什么？翻开下一页之前先花些时间来研究这个错误消息。

启用CGI跟踪来帮助解决错误

CGI标准指出服务器端程序（你的CGI脚本）生成的任何输出都会由Web服务器捕获并发送到等待的Web浏览器。具体来说，会捕获发送到STDOUT（标准输出）的所有内容。

你的CGI脚本产生一个异常时，Python将把错误消息显示在STDERR（标准错误输出）上。CGI机制会忽略这个输出，因为它想要的只是CGI脚本的标准输出。

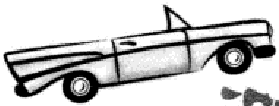


如果Web应用已经部署，这种行为是合适的，不过如果尚在开发就不一样了。要是能在浏览器窗口看到异常的详细信息，而不必不停地跳转到Web服务器的日志屏幕查看，这会很有用。

嗯……你猜怎么样？Python的标准库提供了一个CGI跟踪模块（名为cgitb），启用这个模块时，会在Web浏览器上显示详细的错误消息。这些消息可以帮助你找出CGI中哪里出了问题。改正错误而且CGI正常工作后，再关掉CGI跟踪：

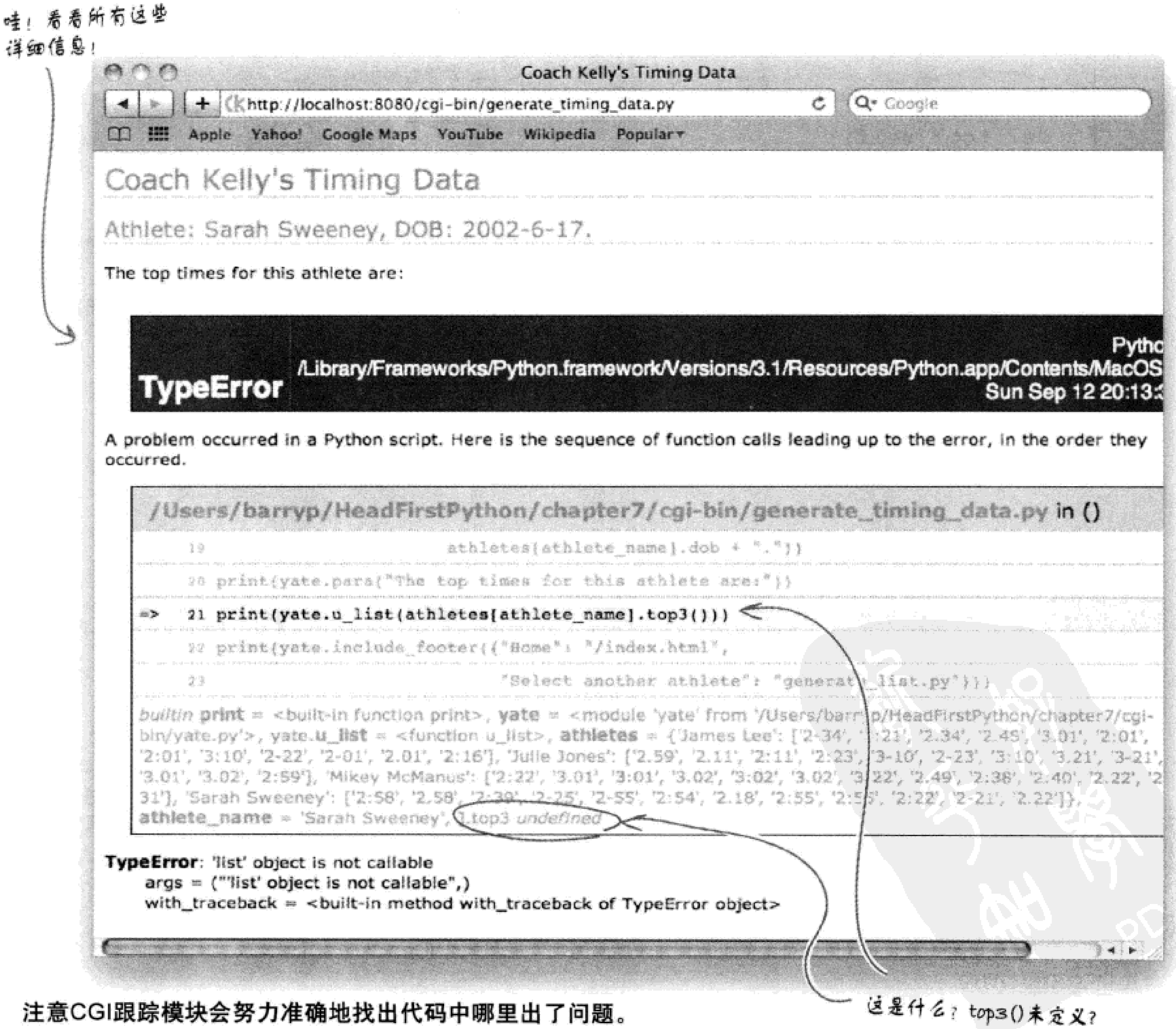
```
import cgitb
cgitb.enable()
```

在CGI脚本最前面增加这两行代码，启用Python的CGI跟踪技术。



测试驱动

在generate_timing_data.py CGI脚本最前面增加两行CGI跟踪代码。在Web浏览器中按“后退”按钮，再次按下“选择”按钮。下面来看这一次会发生什么。



注意CGI跟踪模块会努力准确地找出代码中哪里出了问题。

一个小改变会让一切大不同

CGI跟踪输出指出AthleteList代码中top3()方法的使用有一个错误。

快速查看AthleteList类的代码可以发现错误的原因：top3()方法被重新指定为一个类属性。

```
@property
def top3(self):
    return(sorted(set([self.sanitize(t) for t in self]))[0:3])
```

这个修饰符允许访问"top3()"返回的数据时把它看作是一个类属性。

由于使用了@property修饰符，在类用户看来top3()方法就像是一个属性。所以，不应这样调用top3()方法：

```
print(yate.u_list(athletes[athlete_name].top3()))
```

方法调用总要有括号……

要把top3()方法看作是一个类属性，应当这样调用：

```
print(yate.u_list(athletes[athlete_name].top3))
```

……除非这个方法声明为"@property"，在这种情况下不再需要括号。

这是一个很小的改变，但确实很重要

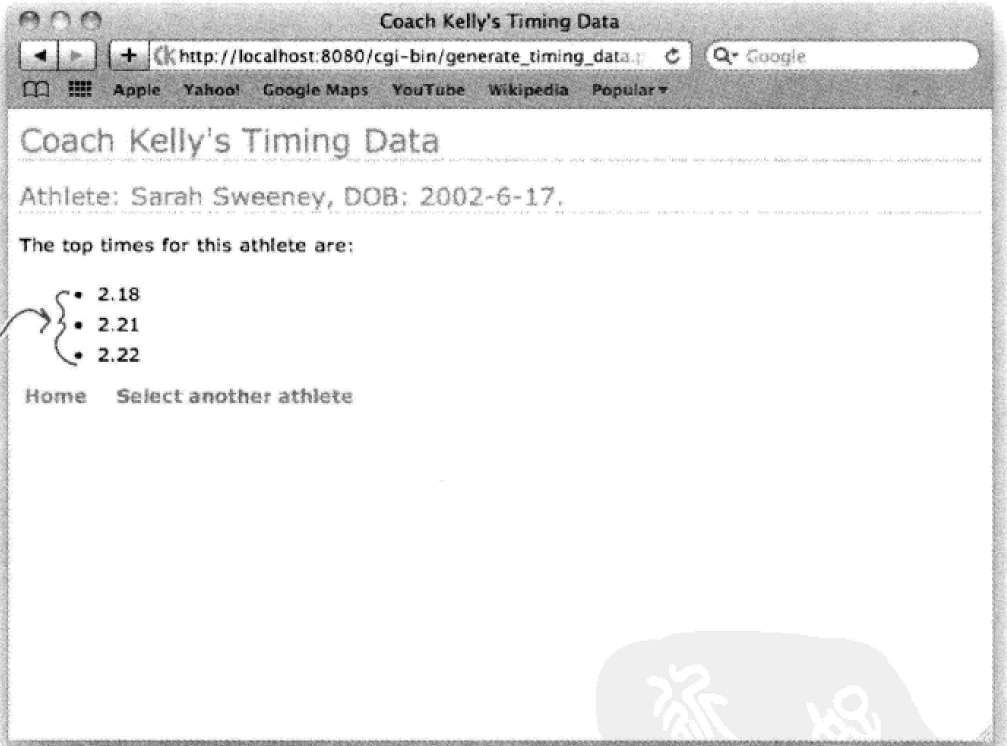
如果对类的使用方式做出改变，你要当心，一定要考虑到这个改变会对现有的程序带来什么影响，这包括你自己的程序以及其他人写的程序。

当然现在只有你一个人在使用AthleteList类，所以修正这个问题还不算太困难。不过请想想看，如果数以千计的程序员都在使用你的代码而且非常依赖你的代码会怎么样呢……

下面来修正你的CGI脚本，再试试看。

测试驱动

对代码做这个小小的修改，去掉top3()方法调用后面的括号，在Web浏览器中按下“后退”按钮，最后再按一次“选择”按钮。



哈哈！这一次屏幕上显示了所选的选手的数据。很不错，是吧？

既然你已经解决了这个问题，一定要关闭CGI跟踪。

there are no Dumb Questions

问： 如果教练招收新的选手会怎么样呢？

答： Kelly教练所要做的就是创建一个与其他文本文件类似的新文本文件，下一次运行你的Web应用时，也就是有人点击主页的“timing data”（计时数据）超链接时，它会动态地加入这个新选手并处理其余的工作。

问： 服务器的数据怎么放在pickle中，难道不该放在数据库中吗？那样肯定更好一些，是不是？

答： 对于这里的情况使用数据库可能有些大材小用，不过将来确实值得考虑。

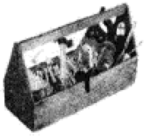
你的Web应用妙极了！



通过把程序移到Web上，现在Kelly教练不费吹灰之力就可以与他的选手们共享数据，不仅如此，任何需要访问这个数据的人都可以分享。

通过遵循MVC模式并使用CGI，你已经构建了一个Web应用，而且有新的需求时可以很容易地扩展这个应用。

祝贺你！你已经成为一个Web开发人员。



你的Python工具箱

你已经读完了第7章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- "@property"——这是一个修饰符，可以使类方法表现得像是一个类属性。

Web术语

- "Web应用"——在Web上运行的一个程序。
- "Web请求"——从Web浏览器发送到Web服务器。
- "Web响应"——从Web服务器发送到Web浏览器，作为对Web请求的响应。
- "CGI"——通用网关接口（Common Gateway Interface），允许Web服务器运行一个服务器端程序。

BULLET POINTS

- 模型－视图－控制器（Model-View-Controller）模式允许你采用一种可维护的方式设计和构建一个Web应用。
- 模型存储Web应用的数据。
- 视图显示Web应用的用户界面。
- 控制器将所有代码与编程逻辑"粘合"在一起。
- 标准库`string`模块包括一个名为`Template`的类，它支持简单的字符串替换。
- 标准库`http.server`模块可以用来在Python中建立一个简单的Web服务器。
- 标准库CGI模块对编写CGI脚本提供了支持。
- 标准库`glob`模块非常适合处理文件名列表。
- 在Linux和Mac OS X上可以使用`chmod +x`命令设置可执行权限位。
- 启用标准库`cgitb`模块时，允许在浏览器中查看CGI编码错误。
- CGI代码中可以使用`cgitb.enable()`打开CGI跟踪。
- 可以使用`cgi.FieldStorage()`访问作为Web请求一部分发送给Web服务器的数据，数据将作为一个Python字典。

8 移动应用开发

小设备



数据放在Web上就像打开了潘朵拉的盒子。

不仅任何人可以从任何地方与你的Web应用交互，而且越来越多的人在通过各种各样的计算设备访问你的Web应用，比如PC、笔记本电脑、平板电脑、掌上电脑，甚至移动电话。现在你不仅需要支持和考虑与Web应用交互的人，还要考虑机器人，也就是能自动完成Web交互的小程序，它们通常只想得到你的数据，并不需要对人类友好的HTML。这一章将在Kelly教练的移动电话上利用Python编写一个应用，来访问Web应用的数据。

世界越来越小

Kelly教练还在每天使用他的Web应用，不过他的新智能手机有点问题。

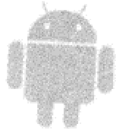


不光只有桌面计算机。

用户有可能想用桌面计算机或笔记本电脑以外的某种设备访问你的Web应用，谁知道呢！

如今我们要面对一个丰富多彩的计算环境。

Kelly教练在使用Android



教练有一个很棒的新智能手机，它运行的是Google的Android操作系统。当然，可以看到，在手机的3英寸屏幕上，这个Web应用确实太小了，根本无法使用。





很显然，教练需要访问他的数据，想在他的手机上运行这个Web应用……不过如果不通过手机的浏览器，最好用什么办法？

Sharpen your pencil

在你的桌面计算机（或手机）上打开Web浏览器，在你最喜欢的搜索引擎中输入“Python for Android”。在下面给出的空白处记录搜索结果中最有价值的网站：

....

....

Sharpen your pencil Solution

在你的桌面计算机（或手机）上打开Web浏览器，在你最喜欢的搜索引擎中输入“Python for Android”。在下面给出的空白处记录搜索结果中最有价值的网站：

你找到的是这个吗？

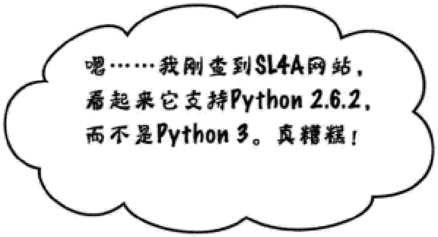
http://code.google.com/p/android-scripting（SL4A项目的主页）。

在教练的智能手机上运行Python

在网上快速搜索你会有一个惊喜的发现：Python可以在Android上运行。

至少有一个Python版本可以在Android上运行。有关技术由一个名为Scripting Layer for Android (SL4A)的项目提供，它允许在任何Android设备上运行Python。不过有一个问题。







是的，SL4A支持Python 2，而不是Python 3。

这本书参照的是Python 3（目前Python最好的版本），不过它的优越性是有代价的，它缺乏向后兼容性。版本3中有些特性在版本2中不能正常工作，反之亦然。

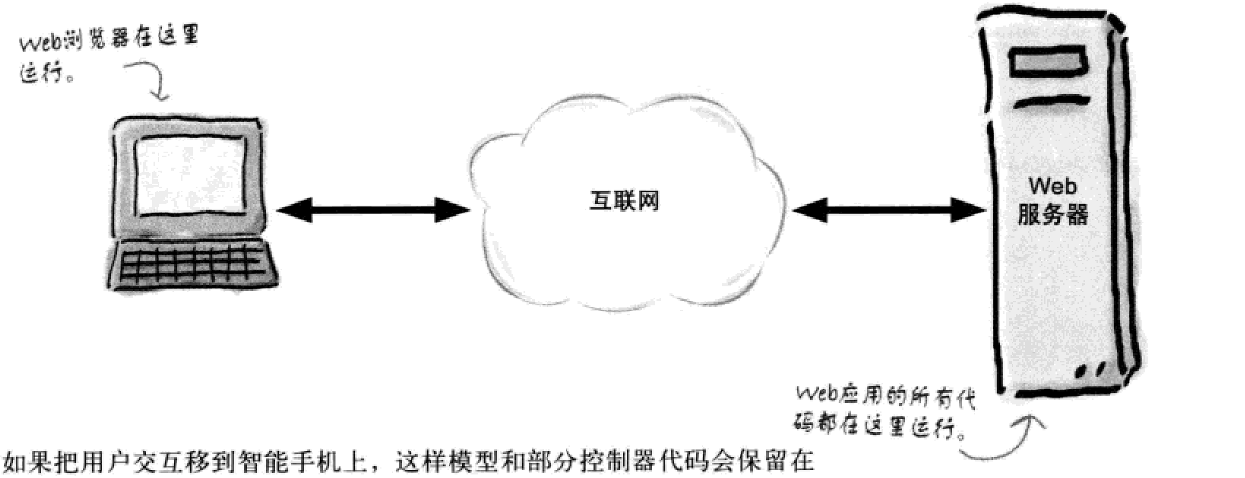
这是不是意味着我们无法继续了？

不用担心Python 2

Android支持Python 2，而这本书介绍的是Python 3，但这一点根本无需担心。Python 2也是Python，Python 2和Python 3之间的差别很容易管理。

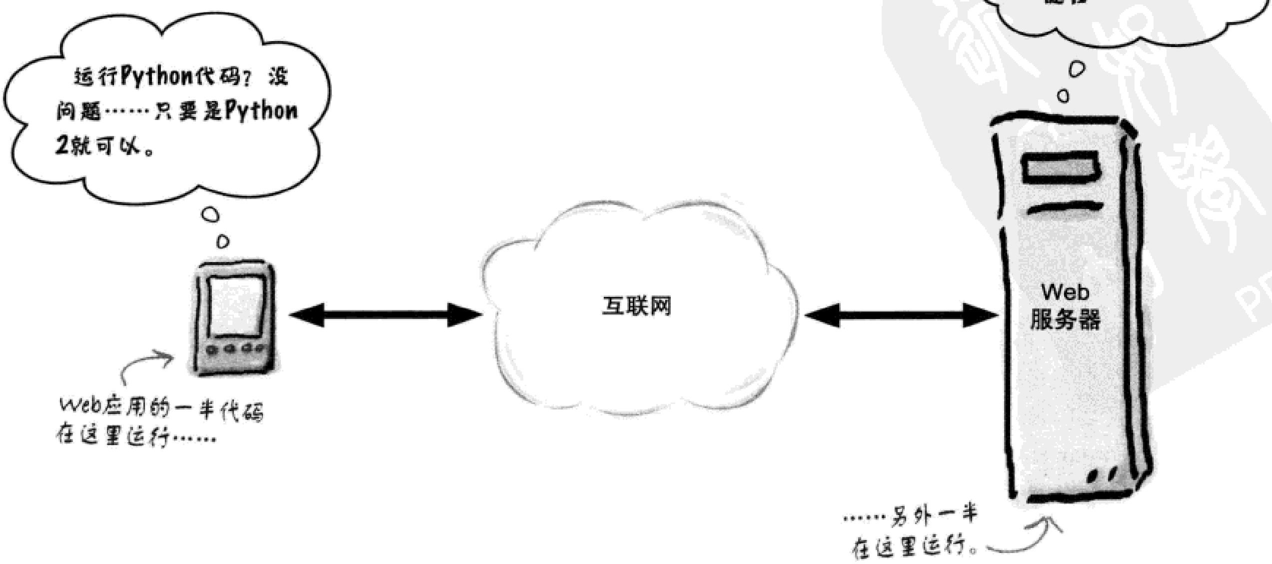
先来考虑你的Web应用。现在，模型、视图和控制器代码都放在Web服务器上，它运行的是Python 3。





如果把用户交互移到智能手机上，这样模型和部分控制器代码会保留在服务器上（仍然运行Python 3），而视图代码和其余的控制器代码将移到智能手机上，就需要重写这些代码从而在Python 2上运行。





建立开发环境

可以理解，教练不会让你一直拿着他的手机，直到你开发出他想要的应用。好在，Google提供了一个跨平台的**Android模拟器**，允许根据需要完成手机应用开发（即使你没有任何硬件）。

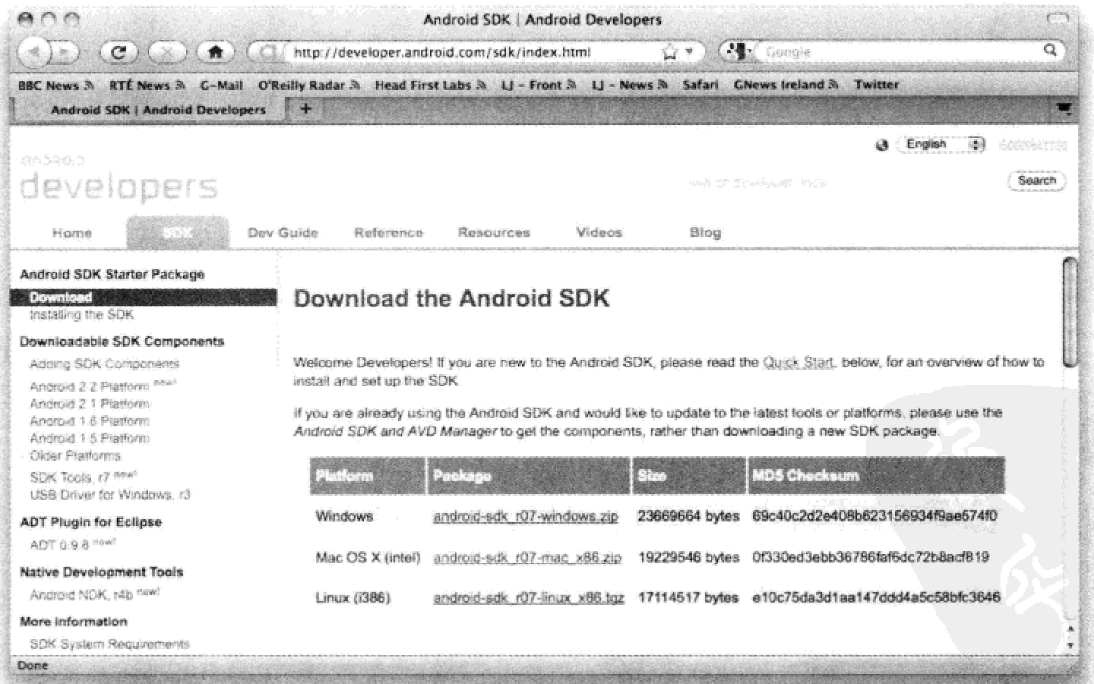
动手做！

按照这些说明，确保在你的计算机上正确地建立Android开发环境。

下载软件开发包(SDK)

下面开始面向Android开发应用。访问以下网站，为你的计算机和操作系统下载合适的SDK：

http://developer.android.com/sdk/index.html



Android SDK网站。

[redacted]书时的Android SDK下载页面。你看到的可能与此稍有不同。不用担心：只需下载SDK的最新版本就可以了。

尽管这个网站看起来像是要求安装Eclipse，但实际上要运行Android模拟器并不需要安装Eclipse。不过，确实需要安装一个Java运行时环境（Java Runtime Environment，JRE）。如果不能确定是否已安装JRE，也不用担心，倘若发现没有安装Java，Android模拟器会给出安装JRE的最佳建议。

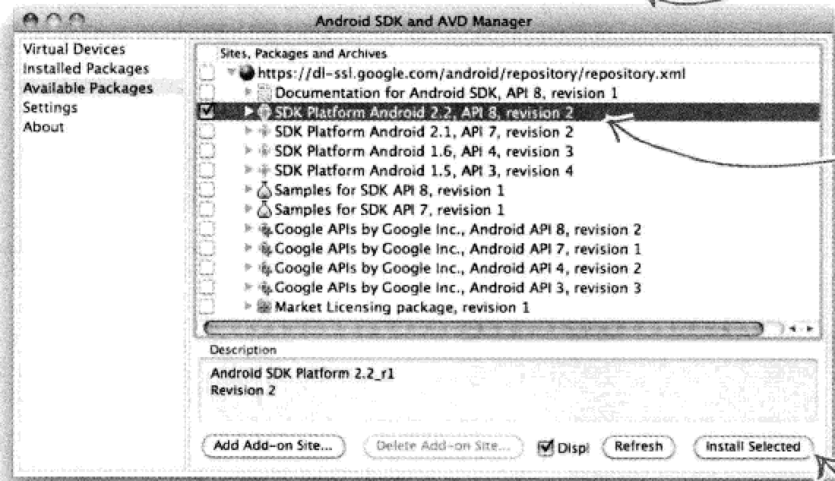
配置SDK和模拟器

要配置SDK和模拟器需要做两件事情：增加一个Android平台，另外创建一个Android虚拟设备（也称为AVD）。

增加一个Android平台

教练的手机上运行的是Android 2.2，所以下面增加一个2.2平台来模拟这个环境。打开Android SDK and AVD Manager（Android SDK和AVD管理器）工具，选择Available Packages（可用包），然后选择安装2.2。

Android下载包包含一个名为"tools"的文件夹。在这个文件夹中运行"android"程序。

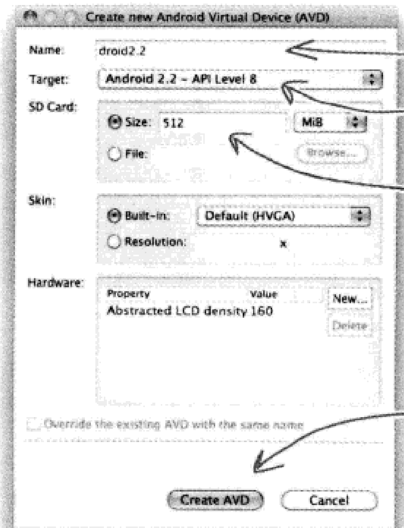


这正是你需要的SDK版本。

安装可能需要用一、两分钟时间，这取决于你的网络连接的速度。

创建一个新的Android虚拟设备（AVD）

下载并安装2.2平台后，创建一个新的Android虚拟设备。



为AVD指定一个名，并选择一个目标。

选择虚拟SDcard的大小：512就足够了。

点击"Create AVD"（创建AVD）。

AVD是一个模拟的Android手机。

安装和配置Android脚本环境

有了模拟器，下面使用AVD管理器启动你的2.2设备。点击模拟器的浏览器（小地球图标），导航到以下Web地址：

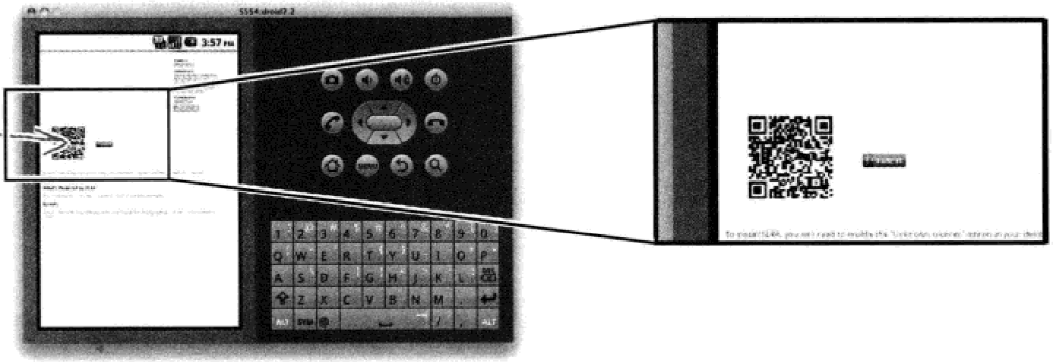
这些要求同样适用于“真正的”手机。只是要确保启用“未知来源”，允许下载非市场应用。

http://code.google.com/p/android-scripting

点击靠近页面下方的“方块”条形码：

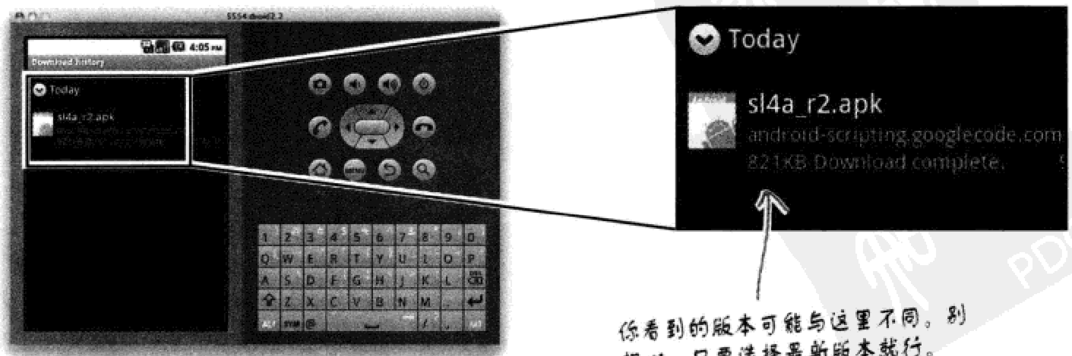
如果你的模拟器要用一、两分钟才能启动，不用担心。模拟器往往比真正的手机慢……

在模拟器上轻击“方块”条形码启动SL4A下载。



下载完成后，选择模拟器的Menu（菜单）按钮，再选择“More”（更多）→“Downloads”（下载），然后点击sl4 _ r2.apk文件，在模拟器上安装SL4A包。安装完成时，点击Done（完成）。

你可用的版本可能有所不同，不过不用担心，只需要下载最新的版本。



你看到的版本可能与这里不同。别担心：只要选择最新版本就行。

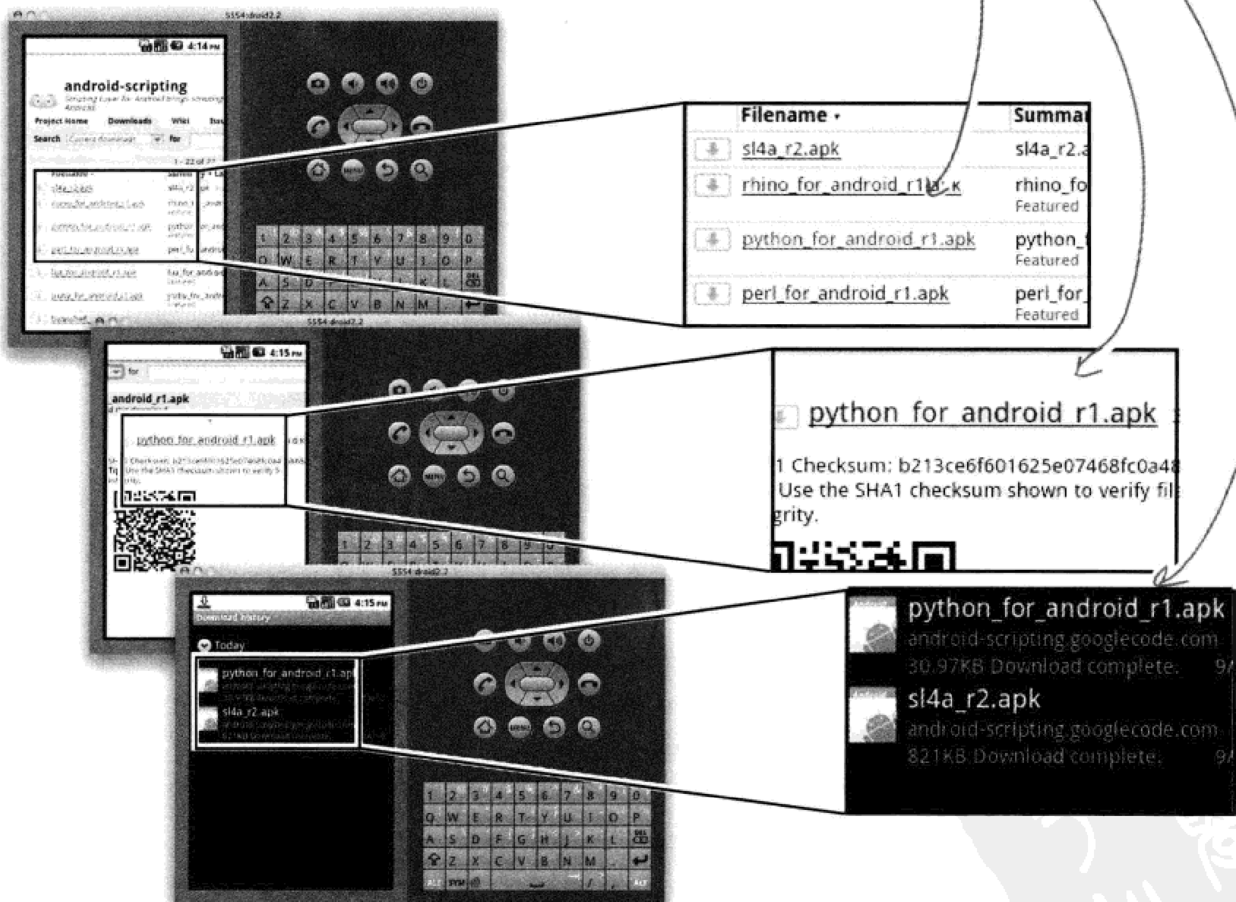
为SL4A安装增加Python

返回到模拟器的Web浏览器，两次点击屏幕进行放大，选择Downloads（下载）标签页。再做两次点击，然后点击以下链接：

python_for_android_r1.apk

同样的，你看到的版本可能与这里不同，只需选择最新的文件。

点击这个下载链接，再点击包名来下载这个包。选择“Menu”（菜单）→“More”（更多）→“Downloads”（下载），再点击新下载的包。



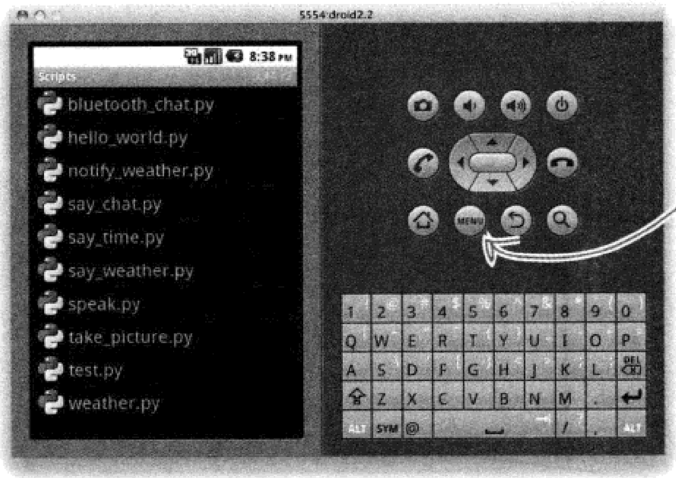
Python for Android应用开始运行。准备好后，点击“Open”（打开）→“Install”（安装）来完成安装。这会下载、解压并安装面向Android的Python支持文件，可能需要几分钟才能完成。完成后，Python 2.6.2和Python for Android都将安装在你的模拟器上可供使用。

最后这一步非常重要。

下面通过一个简单的测试来确认一切正常。

在Android上测试Python

返回到模拟器的主屏，找到一个名为SL4A的应用（已经增加到应用图标列表中）。点击这个应用，显示Python for Android预安装的一组Python脚本。可以点击任何脚本名来执行该脚本：



Watch it!

一定要把SL4A旋转模式设置为自动

第一次运行一个脚本时你的屏幕可能会默认地切换为水平模式。为了修正这一点，请选择Menu（菜单）->Preference（首选项），向下滚动到Rotation mode（旋转模式），将值设置为Automatic（自动）。

尝试Android模拟器

下面是一个包含4行代码的Python脚本，可以创建这个脚本来测试你的安装环境。将这个脚本命名为mydroidtest.py：

导入"android"库，并创建一个新的应用对象实例。

```
import android
app = android.Android()
msg = "Hello from Head First Python on Android"
app.makeToast(msg)
```

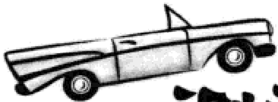
创建一个合适的消息，并在屏幕上显示。

要把脚本传送到模拟器，需要把它复制到模拟器的虚拟SD卡。tools文件夹中有一个名为adb的程序可以提供帮助：

```
tools/adb push mydroidtest.py /sdcard/sl4a/scripts
```

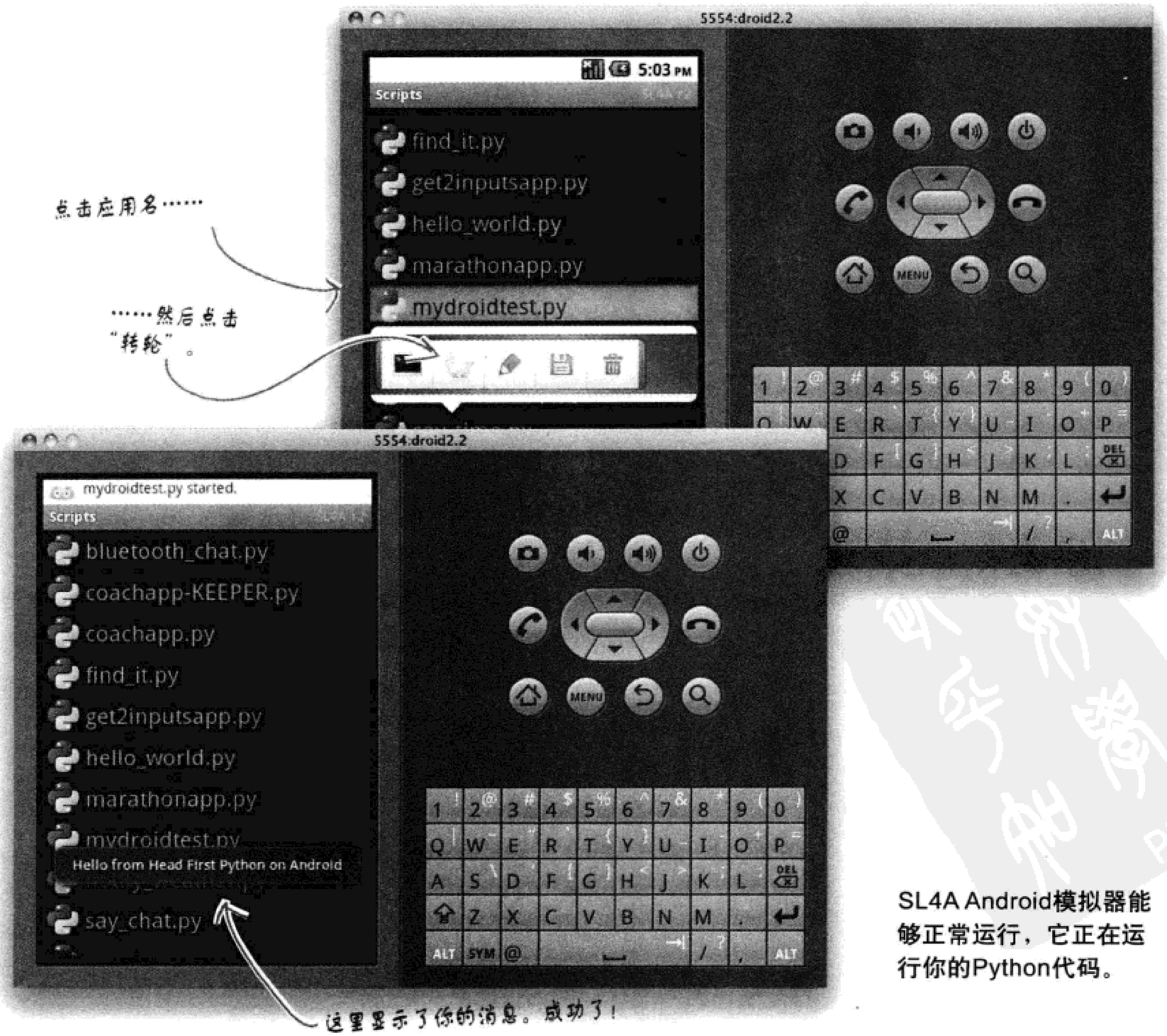
在你的终端窗口执行这个命令，将脚本传输到模拟器。

现在你的脚本应该出现在SL4A可用的脚本列表中了。



测试驱动

下面确保Android环境可以正常运行。打开SL4A应用，轻击你的脚本名来运行脚本，然后从菜单点击“转轮”。



SL4A Android模拟器能够正常运行，它正在运行你的Python代码。

定义应用的需求

下面来想想你的Android应用需要做什么。



Frank: 嗯……首先，视图代码不用生成HTML了，这有点意思。

Jill: 实际上，只需要Web服务器根据请求提供数据，根本不需要那些生成的HTML。

Joe: 哈哈！我知道怎么做了。只需要从服务器把包含所有数据的pickle发送到Android手机就行了。并没有多难，不是吗？

Jill: 可是很遗憾，这会带来很多问题。Python 3使用的pickle格式与Python 2不兼容。你当然能把pickle发送到手机上，但是手机的Python却无法处理pickle中的数据。

Frank: 太糟糕了……那我们该怎么办？传送无格式的数据？

Joe: 嘿，这主意不错：只需要把数据作为一个大字符串发送，再在手机上解析就可以了。听上去是个可行的办法，对吗？

Jill: 不，这会带来灾难，因为你永远不会知道到来的字符串数据会采用什么格式。你需要一种数据交换格式，比如说XML或JSON。

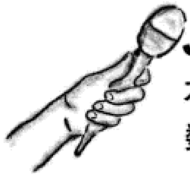
Frank: 嗯……我听说XML使用起来很麻烦……而且对于这么简单的一个应用使用XML可能有些大材小用了吧。JSON是怎么回事？

Joe: 对，当然，我总听人说起JSON。我想Web上的很多不同地方都在用JSON，特别是AJAX。

Frank: 我的天呀……pickle、XML、JSON，现在又来了AJAX……我觉得脑袋都要炸了。

Jill: 别担心，你只需要知道JSON就行了。实际上，你甚至根本不用了解JSON，只要知道怎么用就行。另外，想不到吧？JSON不仅支持Python 2，也支持Python 3……而且格式是兼容的。所以，Web服务器和手机上都可以使用JSON。

Frank & Joe: 太棒了！我们就喜欢这样的技术！



JSON出场

本周访谈：数据交换内幕

Head First:你好，JSON。感谢你愿意和我聊聊。

JSON:没关系。只要力所能及，很乐意效劳。

Head First: 可以说说吗？你到底是什么？

JSON:噢，我只是Web上使用最广泛的数据交换格式之一。需要在互联网上传输数据时，你就可以来找我。当然，你会发现我无处不在。

Head First: 为什么呢？

JSON:嗯……这与我的名字有关。JSON中的“JS”代表“JavaScript”，“ON”代表“对象记法”。明白了吗？

Head First: 呃……我还是不太清楚。

JSON:我就是JavaScript对象记法，这说明哪里都有我。

Head First: 对不起，不过你把我完全搞糊涂了。

JSON: 前两个字母最关键。我是一个JavaScript标准，这说明只要有JavaScript你就能找到我……也就是说这个世界上的每个主流浏览器里都有我。

Head First: 这与Python有什么关系？

JSON:这就要说到后两个字母了。因为我原先设计为只是允许JavaScript数据对象从一个JavaScript程序传输到另一个程序，所以后来得到了扩展，不论使用什么编程语言创建数据，数据对象都能顺利传输。通过使用你喜欢的编程语言所提供的JSON库，就能创建可以交换的数据。如果可以读取一个JSON数据流，还可以在必要时重新创建数据。

Head First: 这么说，我可以得到一个（比如）用Python创建的对象，用JSON把它转换为JSON对象记法，然后把转换后的数据发送给另一个运行着C#程序的计算机，是吗？

JSON:只要C#有一个JSON库，你就可以把Python数据重新创建为C#数据。是不是很棒？

Head First: 确实，听上去很有意思……只是[眨眼]为什么会有人想用C#编程呢？

JSON: [大笑]噢，拜托，别那么小心眼。使用不同的编程语言有很多原因。

Head First: 从某种意义上讲，这正好能解释为什么我们有这么多一流的编程书，比如《Head First C#》、《Head First Java》、《Head First PHP and MySQL》、《Head First Rails》还有《Head First JavaScript》。

JSON: 这算不算个广告？

Head First: 你懂的……我想这也算是吧！[大笑]。

JSON: [笑]。没错，真是在做广告。

Head First: 也是为了共享数据，是不是？

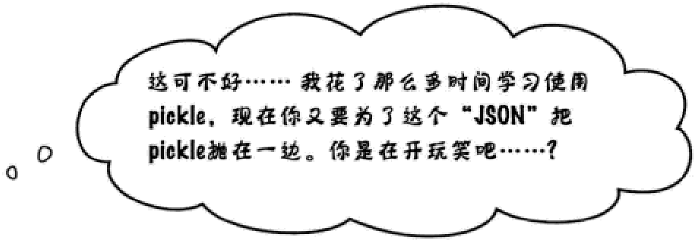
JSON:对！这正是我的重点：如果需要一个易于使用而且与语言无关的数据交换格式，你很难绕开我。

Head First: 不过，既然你名字里有JavaScript，你能做到“与语言无关”吗？

JSON:噢，那只是我的名字而已。他们这样叫我，是因为那个时候我支持的唯一的语言就是JavaScript，后来就这么沿用下来了。

Head First: 这么说他们应该给你换个名字，是吗？

JSON:对，如果我的名字叫“WorksWithEvery-ProgrammingLanguageUnderTheSunIncluding-PythonObjectNotation”（适用Sun之下所有编程语言也包括Python对象记法），听上去是不是大不相同！





你并没有真正“抛弃”pickle。

在这里JSON技术更合适，这有很多原因。首先，这是一种基于文本的格式，所以与Web的工作方式更一致。其次，这是一个标准，在Python 2和Python 3上的工作完全相同，所以这里没有兼容性问题。第三，由于JSON与语言无关，所以完全可以使用其他编程语言编写的Web工具与服务器交互。

如果这里使用pickle，就会失去上述的所有好处。

An IDLE Session

JSON是Python 2和Python 3都已预置的一个成熟的Web标准。JSON API与pickle使用的API并没有太大不同：

```
>>> import json
>>> names = ['John', ['Johnny', 'Jack'], 'Michael', ['Mike', 'Mikey', 'Mick']]
>>> names
['John', ['Johnny', 'Jack'], 'Michael', ['Mike', 'Mikey', 'Mick']]

>>> to_transfer = json.dumps(names)
>>> to_transfer
'["John", ["Johnny", "Jack"], "Michael", ["Mike", "Mikey", "Mick"]]'

>>> from_transfer = json.loads(to_transfer)
>>> from_transfer
['John', ['Johnny', 'Jack'], 'Michael', ['Mike', 'Mikey', 'Mick']]

>>> names
['John', ['Johnny', 'Jack'], 'Michael', ['Mike', 'Mikey', 'Mick']]
```

导入JSON库。

创建一个包含列表的列表（多重列表）。

将Python多重列表转换为一个JSON多重列表。

格式相似，但还是有所不同。

将JSON多重列表转换回Python能理解的格式。

新数据与原来的多重列表完全相同。

Sharpen your pencil

为athletemodel模块增加一个新函数，调用这个函数时，要将选手名列表作为一个字符串返回。

将这个新函数命名为get_names_from_store()。

....

....

....

....

....

....

Sharpen your pencil Solution

为athletemodel模块增加一个新函数，调用这个函数时，要将选手名列表作为一个字符串返回。

将这个新函数命名为get_names_from_store()。

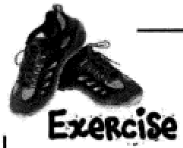
```
def get_names_from_store():
    athletes = get_from_store()
    response = [athletes[each_ath].name for each_ath in athletes]
    return(response)
```

从pickle得到所有数据。

从数据抽取一个选手名列表。

将列表返回给调用者。

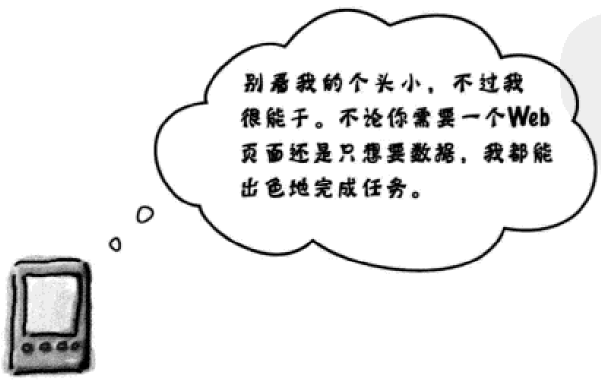




既然已经编写了新函数并增加到athletemodel模块，下面创建一个新的CGI脚本，调用这个脚本时，将从get _ names _ from _ store()函数将数据作为一个JSON数据流返回给Web请求者。

这个新脚本命名为cgi-bin/generate _ names.py。

提示：Content-type要使用application/json。



Exercise Solution

既然已经编写了新函数并增加到athletemodel模块，下面创建一个新的CGI脚本，调用这个脚本时，将从get_names_from_store()函数将数据作为一个JSON数据流返回给Web请求者。

这个新脚本命名为cgi-bin/generate_names.py。

```
#! /usr/local/bin/python3

import json
import athletemodel
import yate

names = athletemodel.get_names_from_store()

print(yate.start_response('application/json'))
print(json.dumps(sorted(names)))
```

如果在Linux或Mac OS X上运行，不要忘记最前面这行“神奇”的代码。

完成导入。

从模型得到数据。

首先是适当的“Content-type:”行。

对“names”排序，然后转换为JSON，并发送到STDOUT。

Watch it!

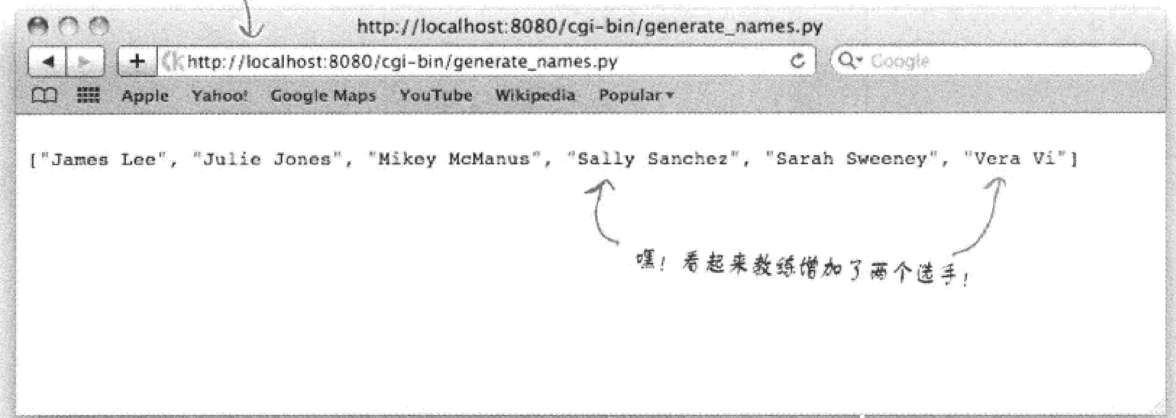
注意测试JSON生成的CGI代码。

测试JSON生成的CGI脚本时你看到的行为可能存在差异，这取决于你使用的Web浏览器。例如，Firefox可能会试图下载生成的数据，而不是在屏幕上显示出来。

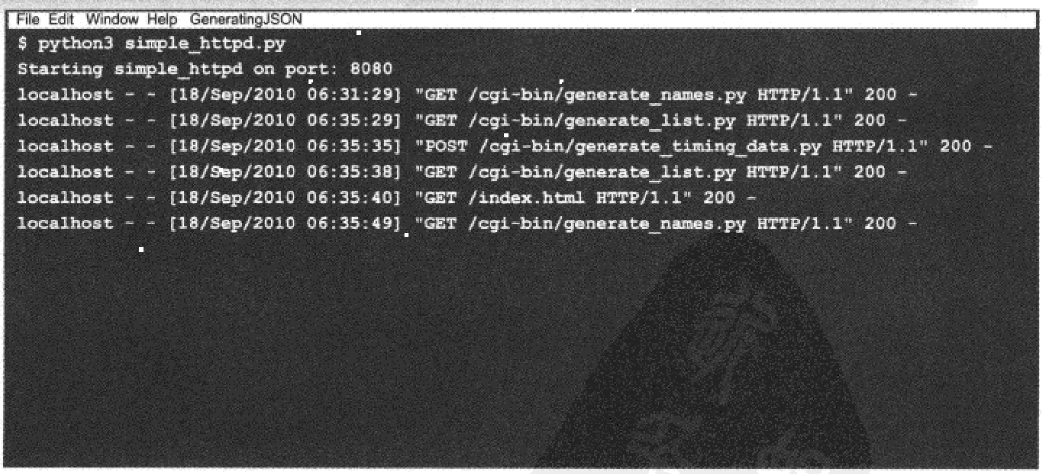
测试驱动

如果未运行Web服务器，现在启动Web服务器，一定要用chmod +x cgi-bin/generate_names.py 命令设置可执行权限位（如果在Linux或Mac OS X上）。准备就绪后，用你最喜欢的Web浏览器尝试运行这个新CGI脚本。

在浏览器的地址栏输入CGI的Web地址。



嘿！看起来数据增加了两个选手！



Web服务器的日志信息确认了CGI确实得到执行。

成功了！

现在你要做的就是在一个Python脚本中让Android模拟器请求数据，并在智能手机屏幕上显示名字列表。这会很难吗？

SL4A Android API

SL4A技术为底层Android API提供了一个高层API，SL4A API的文档见联机API参考：

http://code.google.com/p/android-scripting/wiki/ApiReference

回顾前面的代码，其中展示了一个很小的Android SL4A应用：

导入“android”库，并创建一个新的应用对象实例。

创建一个适当的消息，并在屏幕上显示。

```
import android
app = android.Android()
msg = "Hello from Head First Python on Android"
app.makeToast(msg)
```

通过6个Android　API调用，可以在对话框中创建一个选项列表，并创建相应的正负按钮，用来指示用户所做的选择。注意各个Android“对话框”API调用在屏幕上会得到怎样的输出。

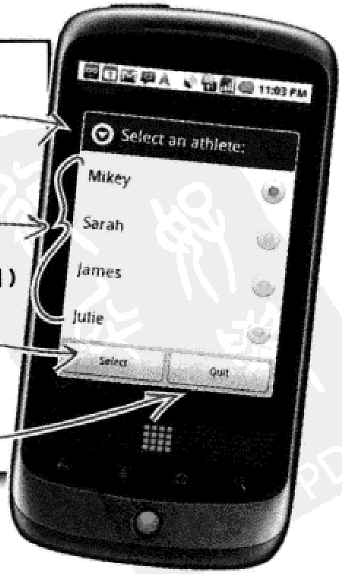
总是首先有一个import。

创建一个Android应用对象。

```
import android

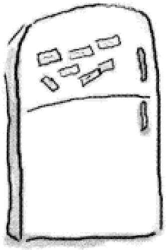
app = android.Android()

app.dialogCreateAlert("Select an athlete:")
app.dialogSetSingleChoiceItems(['Mikey', 'Sarah', 'James', 'Julie'])
app.dialogSetPositiveButtonText("Select")
app.dialogSetNegativeButtonText("Quit")
app.dialogShow()
resp = app.dialogGetResponse().result
```



在手机上显示对话框。

等待用户的响应。



Android代码磁贴

下面的程序代码将向Web服务器查询名字列表，作为一个JSON数组返回，并在智能手机上显示这个列表。现在的麻烦是，程序的后半部分只是一堆乱七八糟的代码磁贴，堆在屏幕下面。你的任务是重新摆放这些磁贴完成这个程序。

```
import android
import json
import time

from urllib import urlencode
from urllib2 import urlopen

hello_msg     = "Welcome to Coach Kelly's Timing App"
list_title    = 'Here is your list of athletes:'
quit_msg      = "Quitting Coach Kelly's App."
web_server    = 'http://192.168.1.33:8080'
get_names_cgi = '/cgi-bin/generate_names.py'

def send_to_server(url, post_data=None):
    if post_data:
        page = urlopen(url, urlencode(post_data))
    else:
        page = urlopen(url)
    return(page.read().decode("utf8"))
```

与以往一样，首先导入库……这些库可以提供Web客户端功能。

这个程序的所有消息都放在一起。

把这个地址改为你的Web服务器所在的Web地址。

在Web服务器上运行的CGI脚本名。

这个函数取一个Web地址（url）和一些可选数据（post_data），它将向Web服务器发送一个Web请求。Web响应返回给调用者。

这个代码乱七八糟……你能把它摆好吗？

```
athlete_names = sorted(json.loads(send_to_server(web_server + get_names_cgi)))
```

```
status_update(quit_msg)
```

```
resp = app.dialogGetResponse().result
```

```
app.dialogShow()
```

```
app.dialogCreateAlert(list_title)
```

```
status_update(hello_msg)
```

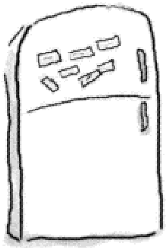
```
def status_update(msg, how_long=2):
    app.makeToast(msg)
    time.sleep(how_long)
```

```
app.dialogSetPositiveButtonText('Select')
```

```
app = android.Android()
```

```
app.dialogSetNegativeButtonText('Quit')
```

```
app.dialogSetSingleChoiceItems(athlete_names)
```



Android代码磁贴答案

下面的程序代码将向Web服务器查询名字列表，作为一个JSON数组返回，并在智能手机上显示这个列表。现在的麻烦是，程序的后半部分只是一堆乱七八糟的代码磁贴，堆在屏幕下面。你的任务是重新摆放这些磁贴完成这个程序。

```
import android
import json
import time
from urllib import urlencode
from urllib2 import urlopen

hello_msg     = "Welcome to Coach Kelly's Timing App"
list_title    = 'Here is your list of athletes:'
quit_msg      = "Quitting Coach Kelly's App."
web_server    = 'http://192.168.1.33:8080'
get_names_cgi = '/cgi-bin/generate_names.py'

def send_to_server(url, post_data=None):
    if post_data:
        page = urlopen(url, urlencode(post_data))
    else:
        page = urlopen(url)
    return(page.read().decode("utf8"))

app = android.Android()

def status_update(msg, how_long=2):
    app.makeToast(msg)
    time.sleep(how_long)

status_update(hello_msg)

athlete_names = sorted(json.loads(send_to_server(web_server + get_names_cgi)))

app.dialogCreateAlert(list_title)
app.dialogSetSingleChoiceItems(athlete_names)
app.dialogSetPositiveButtonText('Select')
app.dialogSetNegativeButtonText('Quit')
app.dialogShow()

resp = app.dialogGetResponse().result

status_update(quit_msg)
```

创建一个Android应用对象。

这是一个小函数，用来在手机上显示简短的消息。

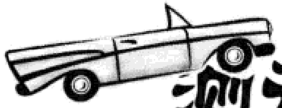
把Web请求发送到服务器，然后将JSON响应转换为一个有序列表。

显示欢迎消息。

由选手名列表创建一个包括两个按钮的对话框。

等待用户点击一个按钮，然后把结果赋给"resp"。

显示退出消息。



测试驱动

应该记得（现在）你的Android Python脚本要在模拟器中运行，而不是在IDLE中运行。所以使用tools/adb程序把程序复制到模拟器。将程序命名为coachapp.py。复制完代码后，在模拟器上启动SL4A，然后点击脚本名。



看起来很不错！你的应用已经与Web服务器通信，请求并接收了选手名列表，并在模拟器上显示了这个列表。



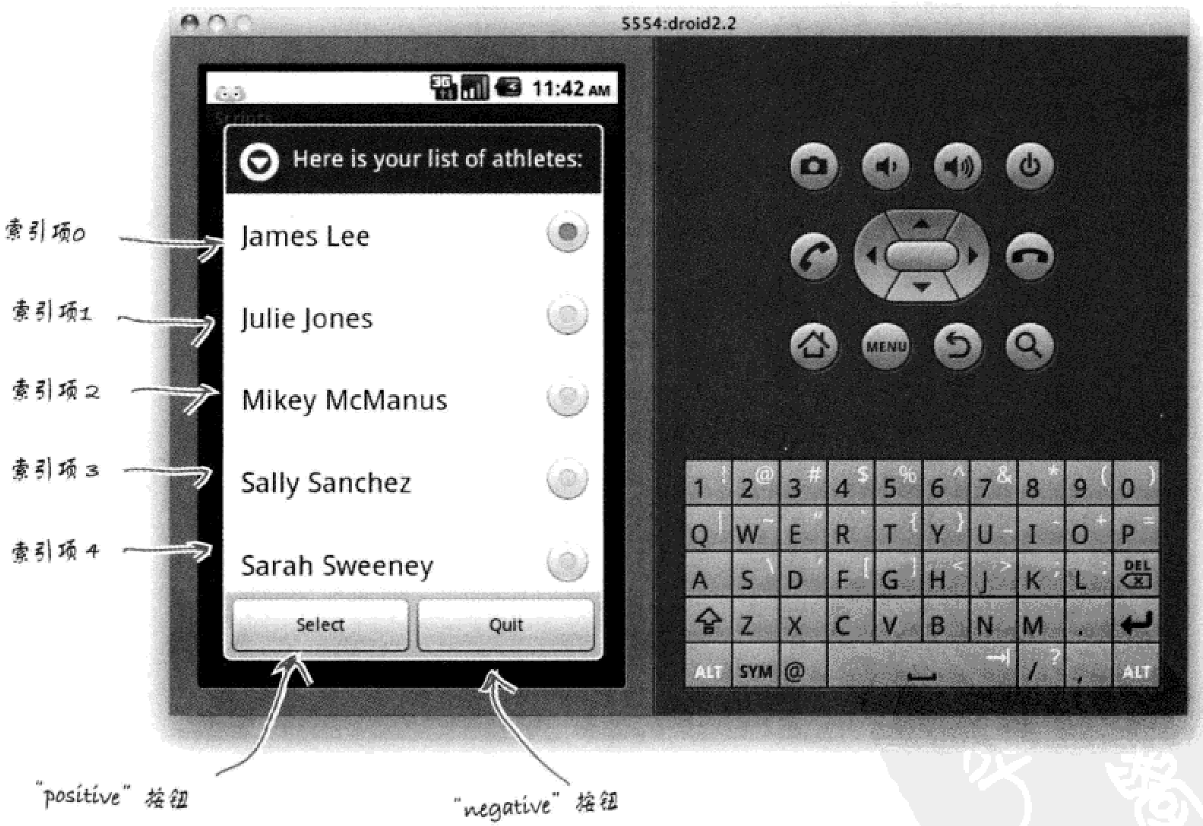
如果你的应用未能运行，不要慌。检查你的代码有没有键入错误。

在SL4A中点击“转轮”左边的小终端图标，再在Python终端中运行你的应用。如果代码产生一个错误，你会看到模拟器的屏幕上会显示一些消息，从中可以了解哪里出了问题。

在Android上选择列表

用户点击一个按钮时，如果点击的是第一个按钮，dialogGetResponse()调用的“结果”会设置为positive（正），如果点击的是第二个按钮，则会设置为negative（负）。在你的代码中，可以检查resp的值，这是一个字典，它的键会设置为positive或negative。

下一个`dialogGetSelectedItems()`调用会返回所选列表项的索引值。



因此……如果点击了positive按钮，可以索引选手名列表来查看从显示的列表中选择了哪个选手。然后使用send_to_server()函数将所选的名字发送到Web服务器，来请求这个选手的其余数据。

可以在下一个版本的代码中使用这个行为。

Sharpen your pencil

❶ 假设有一个名为cgi-bin/generate_data.py的CGI脚本，调用这个脚本时，将从服务器请求一个指定选手的数据。

提供相应代码（包括一个thensend_to_server()函数调用），来实现这个功能：

❷ 另外，编写所需的代码在一个Android对话框中显示从服务器返回的时间列表。

提示：使用Android API的dialogSetItems()方法，为对话框增加一个项列表。另外，要记住通过互联网传送到的数据会使用JSON格式。

Sharpen your pencil Solution

❶ 假设有一个名为cgi-bin/generate_data.py的CGI脚本，调用这个脚本时，将从服务器请求一个指定选手的数据。

提供相应代码（包括一个thensend_to_server()函数调用），来实现这个功能：

提供要运行的CGI的名。

将请求发送到web服务器，并提供选手名。

```
get_data_cgi = '/cgi-bin/generate_data.py'
send_to_server(web_server + get_data_cgi, {'which_athlete': which_athlete})
```

包含数据。

❷ 另外，编写所需的代码在一个Android对话框中显示从服务器返回的时间列表。

按下了哪个按钮？

用户点击"positive"按钮时……得到所选的索引值。

索引值对应对话框返回的结果列表的第一个元素。

使用这个索引值查找选手名。

向服务器发送一个新的web请求，获取这个选手的数据。

动态地创建对话框的标题。

用户这一次只需看到数据，所以需要使用"dialogSetItems()"。

设置按钮的文本。

等待用户点击。

```
if resp['which'] in ('positive'):
    selected_athlete = app.dialogGetSelectedItems().result[0]
    which_athlete = athlete_names[selected_athlete]
    athlete = json.loads(send_to_server(web_server + get_data_cgi,
                        {'which_athlete': which_athlete}))
    athlete_title = which_athlete + ' top 3 times:'
    app.dialogCreateAlert(athlete_title)
    app.dialogSetItems(athlete['Top3'])
    app.dialogSetPositiveButtonText('OK')
    app.dialogShow()
    resp = app.dialogGetResponse().result
```

选手数据CGI脚本

下面是cgi-bin/generate_data.py CGI脚本的代码，它取一个Web请求，从模型返回指定选手的数据。

```
#! /usr/local/bin/python3

import cgi
import json
import athletemodel
import yate

athletes = athletemodel.get_from_store()
form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value
print(yate.start_response('application/json'))
print(json.dumps(athletes[athlete_name]))
```

从模型得到所有数据。

处理随请求发送的数据，并抽取选手名。

启动一个Web请求，数据类型为JSON。

在Web响应中包含指定选手的数据，采用JSON格式。

完整的Android应用（暂时）

目前已经对程序做了很多改动。在Android模拟器上测试之前，先用一点时间完整地看看你的代码：

```
import android
import json
import time

from urllib import urlencode
from urllib2 import urlopen

hello_msg     = "Welcome to Coach Kelly's Timing App"
list_title    = 'Here is your list of athletes:'
quit_msg      = "Quitting Coach Kelly's App."

web_server    = 'http://192.168.1.34:8080'

get_names_cgi = '/cgi-bin/generate_names.py'
get_data_cgi  = '/cgi-bin/generate_data.py'
```

其余代码在下一页上。

```
def send_to_server(url, post_data=None):
    if post_data:
        page = urlopen(url, urlencode(post_data))
    else:
        page = urlopen(url)
    return(page.read().decode("utf8"))

app = android.Android()

def status_update(msg, how_long=2):
    app.makeToast(msg)
    time.sleep(how_long)

status_update(hello_msg)

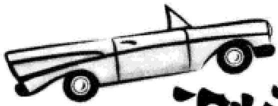
athlete_names = sorted(json.loads(send_to_server(web_server + get_names_cgi)))

app.dialogCreateAlert(list_title)
app.dialogSetSingleChoiceItems(athlete_names)
app.dialogSetPositiveButtonText('Select')
app.dialogSetNegativeButtonText('Quit')
app.dialogShow()
resp = app.dialogGetResponse().result

if resp['which'] in ('positive'):
    selected_athlete = app.dialogGetSelectedItems().result[0]
    which_athlete = athlete_names[selected_athlete]
    athlete = json.loads(send_to_server(web_server + get_data_cgi,
                                        {'which_athlete': which_athlete}))

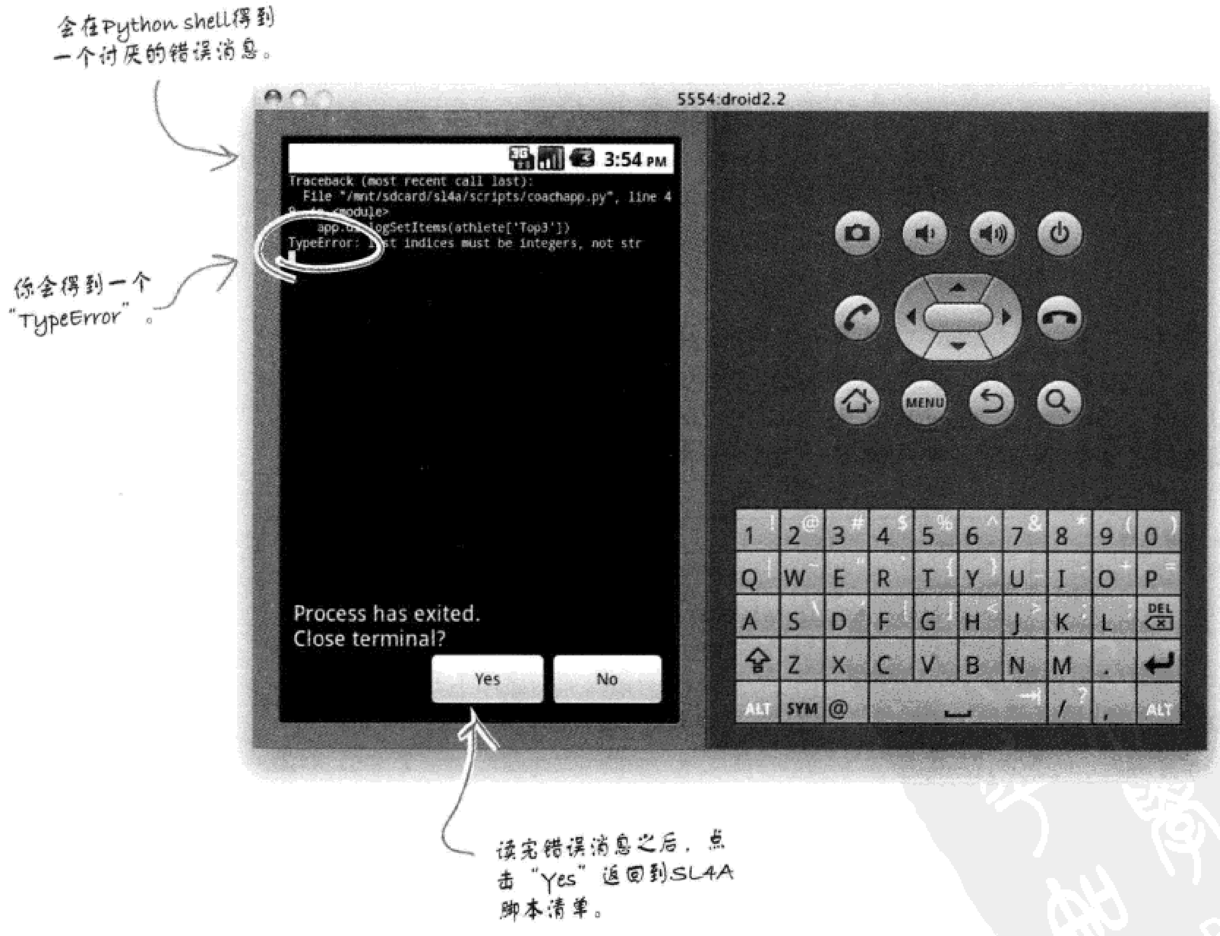
    athlete_title = athlete['Name'] + ' (' + athlete['DOB'] + '), top 3 times:'
    app.dialogCreateAlert(athlete_title)
    app.dialogSetItems(athlete['Top3'])
    app.dialogSetPositiveButtonText('OK')
    app.dialogShow()
    resp = app.dialogGetResponse().result

status_update(quit_msg)
```



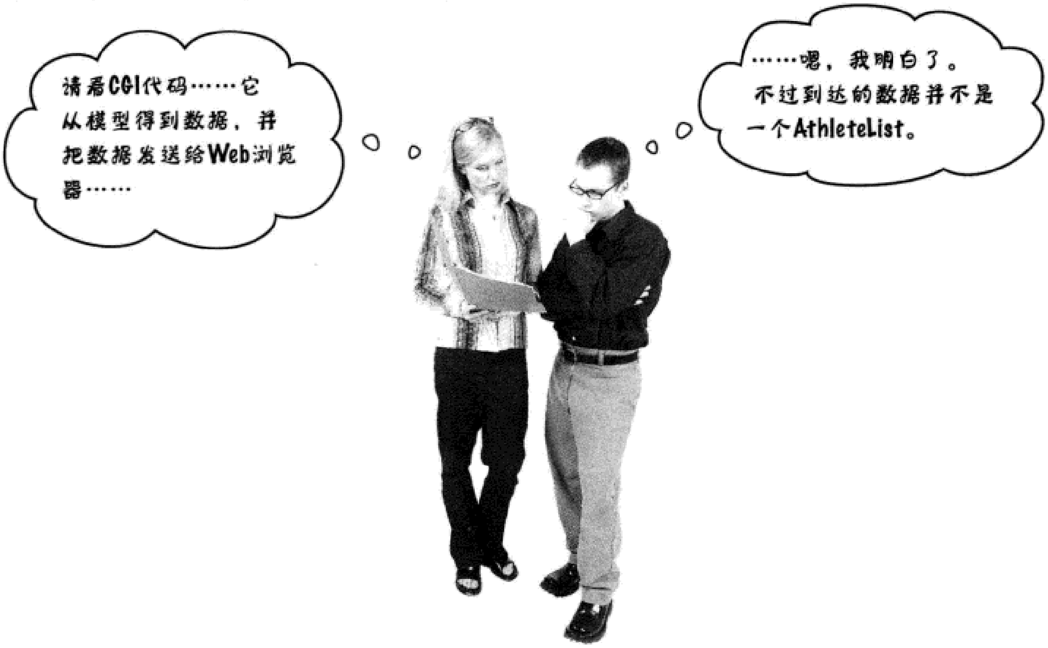
测试驱动

下面来测试应用的最后这个版本。将应用复制到你的模拟器上，并把新的CGI脚本放在Web服务器的cgi-bin文件夹中（要记住，如果需要还应设置可执行权限位）。使用模拟器的Python shell而不是“转轮”运行这个最新应用时会发生什么？



唉呀！ 你的代码有一个TypeError，试图显示所选选手的计时数据时会让你的应用崩溃。你认为为什么会这样？

看起来应该改变数据的类型



下面在CGI脚本中增加一行调试代码来确定到底发生了什么。应该记得，CGI机制默认地会捕获脚本发送到标准输出（STDOUT）的所有输出，所以使用下面的代码将调试消息发送到Web服务器的控制台，这会显示到标准错误输出（STDERR）：

从标准库导入"sys"。

```
import sys

print(json.dumps(athletes[athlete_name]), file=sys.stderr)
```

将输出从"print()"重定向到"stderr"，而不是默认的"stdout"。

再次运行你的应用，当然它还会由于一个TypeError错误而崩溃。不过，如果查看Web服务器的控制台屏幕，你会清楚地看到作为JSON Web响应发送的数据。注意到什么没有？

这是一个选手计时值列表……不过选手名和DOB值在哪里？

```
File Edit Window Help JustWhatsInTheData
$ python3 simple_httpd.py
Starting simple_httpd on port: 8080
192.168.1.33 - - [18/Sep/2010 17:40:04] "GET /cgi-bin/generate_names.py HTTP/1.1" 200 -
192.168.1.33 - - [18/Sep/2010 17:40:08] "POST /cgi-bin/generate_data.py HTTP/1.1" 200 -
["2-44", "3:01", "2.44", "2.55", "2.51", "2:41", "2:41", "3:00", "2-32", "2.11", "2:26"]
```

JSON无法处理你的定制数据类型

pickle非常聪明，可以“腌制”你的定制类，与pickle不同，Python提供的JSON库做不到这一点。这说明，标准库的JSON库只能处理Python的内置类型，而无法处理你的AthleteList对象。

对于这个问题，解决办法很简单：为AthleteList类增加一个方法，将数据转换为一个字典，再把这个字典发回到应用。由于JSON支持Python的字典类型，这种做法应该是可行的。



在AthleteList类中创建一个新方法，命名为to_dict()，这个新方法要把类的属性数据（name、DOB和top3）转换为一个字典。一定要用@property修饰这个新方法，这样一来，对于类用户来说这个方法就像是一个新的属性。

there are no Dumb Questions

问： 这一次使用@property又是出于什么目的？

答： 利用@property修饰符，可以使一个方法对类用户来说就像是一个属性。仔细想想，to_dict()方法并没有以任何方式改变对象数据的状态：它只是把对象的属性数据作为一个字典返回。所以，尽管to_dict()是一个方法，但它表现得更像是一个属性，使用@property修饰符就可以指出这一点。类的用户（也就是其他程序员）并不需要知道他们访问to_dict属性时实际上在运行一个方法。他们所看到的只是一个统一的接口：属性访问类数据，而方法用来管理数据。



在AthleteList类中创建一个新方法，命名为to_dict()，这个新方法要把类的属性数据（name、DOB和top3）转换为一个字典。一定要用@property修饰这个新方法，这样一来，对于类用户来说这个方法就像是一个新的属性。

用"@property"修饰这个新方法。

```
@property
def as_dict(self):
    return({'Name': self.name,
            'DOB': self.dob,
            'Top3': self.top3})
```

创建一个新方法。

返回对象数据属性的一个字典。

没忘记用"self"吧？

动手做！

除了更新AthleteList类代码，还要记得修改cgi-bin/generate-data.py在响应Web请求时返回一个字典，而不是对象实例。

修改代码时，还要调整coachapp.py应用代码，将选手的name和DOB值包含在第二个对话框的标题中。

测试驱动

对AthleteList.py、cgi-bin/generate_data.py和coachapp.py完成修改后，使用adb工具把最新版本的应用复制到模拟器。下面来看现在的情况如何。

应用使用这些代码来响应用户做出的选手选择。



成功了。

你的应用会在屏幕上显示所选选手的前3个时间。是不是很酷？

在真正的手机上运行你的应用

应用已经在模拟器上成功地运行，现在可以在一个真正的手机上试试。这才开始有些意思了。

将代码复制到一个真正的设备时有很多选择：

- 使用蓝牙文件传输。
- 利用USB连接完成文件传输。
- 利用USB使用Android SDK的adb工具。
- 通过Wi-Fi使用文件传输工具。

遗憾的是，具体使用哪种技术（以及哪种技术可行）很大程度上取决于你的手机。

在Head First Labs，我们发现选择最后一种做法最有可能成功：通过Wi-Fi使用文件传输工具。

> **Watch it!**
>
> **这些说明不适用于模拟器**
>
> Android模拟器目前不支持Google的Android Market，而按照这几页上的说明使用AndFTP时需要访问Android Market。

第1步：准备你的计算机。

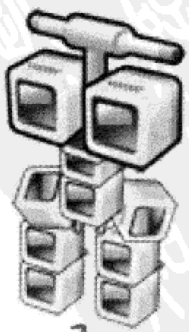
要在你的Android手机和计算机之间安全地传输文件，需要在计算机上运行一个SSH服务器来启用SSH文件传输。如何启用取决于所运行的操作系统：

- Windows：下载一个免费的SSH服务器，这样的免费SSH服务器有很多。
- Mac OS X：启用远程登录。
- Linux：安装和启用OpenSSH服务器。

第2步：在Android手机上安装AndFTP。

使用手机上的Android Market，查找并安装AndFTP应用。这是一个绝妙的工具，允许在计算机和Android手机之间通过FTP、SFTP和FTPS传输文件。

如果计算机上已经运行SSH服务器，要使用AndFTP应用，你可能希望选择SFTP作为应用中的文件传输协议，因为AndFTP默认使用FTP协议。



AndFTP应用是我们最喜欢的工具之一。

下面来看需要做什么。

配置AndFTP

在手机上运行AndFTP之后，将它配置为使用SFTP作为传输协议（类型（Type））与你的计算机（主机名（Hostname））连接。保留端口（Port）、用户名（Username）、密码（Password）和远程目录（Remote dir）不变，不过将本地目录（Local dir）改为/sdcard/sl4a/scripts。



建立连接后，点击AndFTP的Connect（连接）按钮来建立与SSH服务器的连接，提示输入时输入你的用户名和密码。

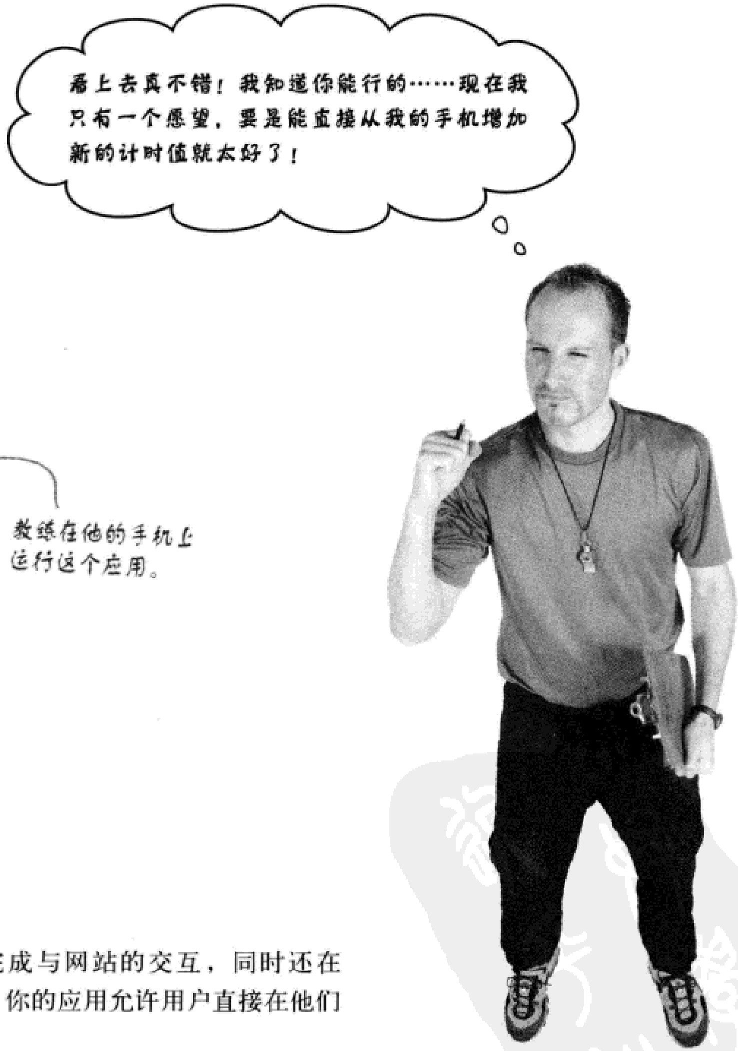
建立了与服务器的连接后，导航到相应的服务器文件夹（其中包含有希望传输到手机的文件），标志要下载的这些文件，并点击Download（下载）按钮。

下载完成时，点击Disconnect（断开连接）终止手机与计算机之间的连接。如果传输一个Python程序，它现在会增加到SL4A中的脚本列表中。



可以让Kelly教练看一看了。

教练对应用大加赞赏





明天会更好！

你已经提出一个解决方案，可以自动完成与网站的交互，同时还在Android手机上提供了一个很现代的界面。你的应用允许用户直接在他们的移动设备上访问Web数据。

服务器代码在Python 3上运行，而Android客户代码运行在Python 2上，这一点并没有带来太大影响。毕竟，它们都是Python代码。

剩下的就是编写一些代码来满足Kelly教练最后的需求，这个任务将在下一章完成。

干得不错。



你的Python工具箱

你已经读完了第8章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- "Python 2"——Python的前一个版本，它与Python 3存在兼容性"问题"（不过不必为此烦恼）。

Android术语

- "SL4A"——全名为Scripting Layer for Android，允许在你的Android设备上运行Python。
- "AVD"——这是一个Android虚拟设备，允许在你的计算机上模拟Android设备。

BULLET POINTS

- JSON库模块允许将Python的内置类型转换为基于文本的JSON数据交换格式。
- 使用json.dumps()可以创建一个Python类型的字符串版本。
- 使用json.loads()可以从一个JSON字符串创建一个Python类型。
- 如果数据使用JSON发送，需要将其Content-Type:设置为application/json。
- urllib和urllib2库模块（都在Python 2中提供）可以用来从一个程序向Web服务器发送编码的数据（使用urlencode()和urlopen()函数）。
- sys模块提供了sys.stdin、sys.stdout和sys.stderr输入流。

9 管理你的数据

处理输入



Web和手机并不只是能很好地显示数据。

它们也是从用户接收输入的绝妙工具。当然，一旦Web应用接收数据，肯定需要把数据放在某个地方。在决定把数据放在哪里时，你的选择往往会对Web应用带来巨大差别。如果选择得当，Web应用将很易于扩展，否则扩展可能很困难。这一章中，你将扩展你的Web应用从Web（通过浏览器或从Android手机）接收数据，另外还将了解并改进后台数据管理服务。

你的选手时间应用已经声名远扬



全国青少年运动员委员会（National Underage Athletics Committee，NUAC）简单查看了你的Android应用，发现这正是他们需要的……几乎可以这么说。

可以从很多方面改进你的Web应用，不过对现在来说，我们要重点关注委员会最迫切的需要：为一个已有的选手数据集中增加新的时间值。

如果向文本文件增加新数据，这种做法是不可取的：全国会有太多教练在增加数据。委员会希望能从Web浏览器或Android手机提供一种对用户友好的增加数据的途径。

你能帮忙吗？

使用表单或对话框接收输入



在Web上，用户可以与Web表单交互并输入数据。用户按下提交按钮时，Web浏览器会收集所有表单数据，并作为Web请求的一部分发送到Web服务器。

在你的Android手机上，可以使用`dialogGetInput()`方法得到用户的输入，然后通过代码模拟Web表单按钮的行为。

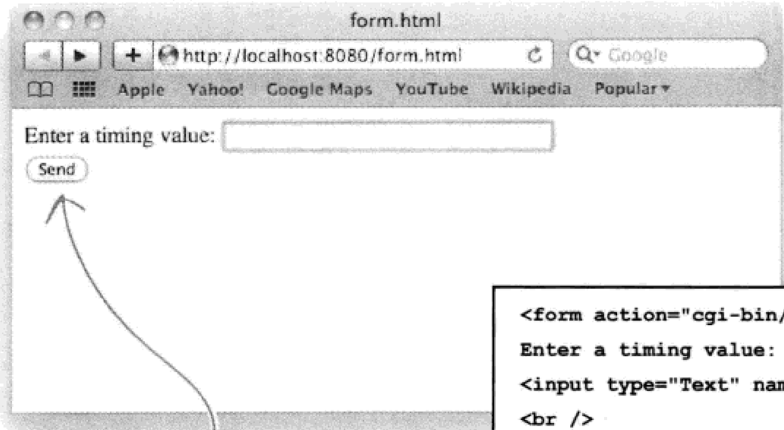
实际上，你已经做过这个工作：查看`coachapp.py`应用中的下面这行代码，这行代码会把选择的选手名发送到你的Web服务器：

在这里将数据包含在Web请求中。

```
athlete = json.loads(send_to_server(web_server + get_data_cgi, {'which_athlete': which_athlete}))
```

创建一个HTML表单模板

下面扩展`yate.py`来支持HTML表单的创建。先看下面这个简单的表单，这里还给出了用来生成这个表单的HTML标记。



点击Send按钮将表单数据提交到Web服务器。

CGI脚本名（表单数据将发送到这个CGI脚本）。

```
<form action="cgi-bin/process-time.py" method="POST">
Enter a timing value:
<input type="Text" name="TimeValue" size=40>
<br />
<input type="Submit" value="Send">
</form>
```

用户点击Send（发送）按钮时，输入区中的所有数据都会作为Web请求的一部分发送到Web服务器。

在你的Web服务器上，可以使用标准库`cgi`模块提供的功能访问CGI数据。

```
import cgi

form = cgi.FieldStorage()
timing_value = form["TimeValue"].value
```

从表单得到作为Web请求一部分发送的数据。

由表单数据访问与"TimeValue"键关联的值。

`CGI`模块将与Web请求关联的数据转换为一个类似字典的对象，可以查询这个对象来抽取你需要的数据。

Exercise

把上一页的HTML表单转变为yate.py模块中的一个模板。

1. 首先创建一个新模板，名为templates/form.html，从而允许对表单的CGI脚本名、方法、输入标记和提交按钮文本指定参数：

....

....

....

2. 创建模板后，为增加到yate.py中的两个函数编写代码。

第一个函数名为create_inputs()，这个函数取一个或多个字符串的列表，为各个字符串分别创建HTML <INPUT>标记（类似前一页上接收TimeValue的<INPUT>标记）。

第二个函数名为do_form()，利用这个练习第一部分完成的模板，并结合create_inputs()生成一个HTML表单。

给定一个<INPUT>标记名列表。

```
def create_inputs(inputs_list):
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    return(html_inputs)
```

将生成的标记返回给调用者。

CGI脚本名和<INPUT>标记名列表是必要参数。

```
def do_form(name, the_inputs, method="POST", text="Submit"):
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    return(form.substitute(cgi_name=name, http_method=method,
                           list_of_inputs=inputs, submit_text=text))
```

HTTP方法和“Submit”按钮的文本有适当的缺省值。

将参数和前面生成的<INPUT>标记替换到模板中来创建表单。

Exercise Solution

把上一页的HTML表单转变为yate.py模块中的一个模板。

❶ 首先创建一个新模板，名为templates/form.html，从而允许对表单的CGI脚本名、方法、输入标记和提交按钮文本指定参数：

```
<form action=$cgi_name method=$http_method>
Enter a timing value: $list_of_inputs<br />
<input type="Submit" value=$submit_text></form>
```

可以为CGI脚本名和相关的HTTP方法指定参数。

<INPUT>标记列表和提交按钮的文本也可以指定参数。

❷ 创建模板后，为增加到yate.py中的两个函数编写代码。

第一个函数名为create_inputs()，这个函数取一个或多个字符串的列表，为各个字符串分别创建HTML <INPUT>标记（类似前一页上接收TimeValue的<INPUT>标记）。

第二个函数名为do_form()，利用这个练习第一部分完成的模板，并结合create_inputs()生成一个HTML表单。

```
def create_inputs(inputs_list):
    html_inputs = ''
    for each_input in inputs_list:
        html_inputs = html_inputs + '<input type="Text" name="' + \
                                    each_input + '" size=40>'
    return(html_inputs)

def do_form(name, the_inputs, method="POST", text="Submit"):
    with open('templates/form.html') as formf:
        form_text = formf.read()
    inputs = create_inputs(the_inputs)
    form = Template(form_text)
    return(form.substitute(cgi_name=name, http_method=method,
                           list_of_inputs=inputs, submit_text=text))
```

这个"延续"字符允许将一个长代码行分为多行。

取各个名字，分别创建一个<INPUT>标记。

从硬盘获得模板。

创建<INPUT>标记列表。

创建一个模板表单。

测试驱动

下面是名为cgi-bin/test-form.py的CGI脚本的代码，它会生成前面的HTML表单。可以看到，这个代码非常简单，而不需要再做更多的说明。

```
#! /usr/local/bin/python3

import yate

print(yate.start_response('text/html'))
print(yate.do_form('add_timing_data.py', ['TimeValue'], text='Send'))
```

首先总是一个CGI响应。

动态创建表单，根据需要提供参数。

使用chmod + x test_form.py设置可执行权限位（如果你的操作系统需要），然后使用浏览器确认这个HTML表单生成代码能正常工作。

在Web浏览器的地址栏输入CGI脚本的URL。



使用浏览器的"查看源代码"菜单项确认所生成的确实是你需要的表单。

生成的HTML表单出现在浏览器窗口中。

太好了。你已经扩展了yate.py，使它能支持创建一个简单的数据输入表单。现在只需要确定数据到达服务器时会发生什么。

数据传送到CGI脚本

除了运行你的Web应用，Web服务器还要把提交的所有表单数据传送给正在等待的CGI脚本。Python的CGI库将数据转换为一个字典，你应该已经知道，它提供了一些方便的办法来访问所提交的数据：

```
import cgi

form = cgi.FieldStorage()
```

所有表单数据已经增加到"form"字典中。

通过Web服务器的环境还可以访问有关Web请求的其他信息。一般地，你并不需要直接访问或使用这些数据。不过，有些情况下，报告其中一些数据可能也很有用。

下面的代码利用了Python的内置支持，可以使用os库查询CGI脚本的环境。这里假设已经由一个友好的Web服务器设置了环境值。注意，环境中的数据在代码中可以作为一个字典来使用。

```
import os
import time
import sys

addr = os.environ['REMOTE_ADDR']
host = os.environ['REMOTE_HOST']
method = os.environ['REQUEST_METHOD']

cur_time = time.asctime(time.localtime())

print(host + ", " + addr + ", " + cur_time + ": " + method, file=sys.stderr)
```

导入的列表中一定要包括"os"库。

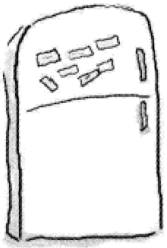
查询3个环境变量，并把它们的值赋至变量。

得到当前时间。

在标准错误输出上显示查询的数据。

下面来利用这一页的两个代码段，将从表单发送的数据记录到Web服务器的控制台。一旦确认数据能够"原封不动"地到达Web服务器，可以进一步扩展代码，将接收到的数据存储在模型中。

下面编写一个CGI脚本来显示表单的数据。



CGI磁贴

你需要一个新的CGI脚本，名为add_timing_data.py，它要处理来自表单的数据，并把这些数据显示在Web服务器的控制台屏幕上。这个CGI需要查询环境，将所记录的数据显示在一行上。所有代码都已经有了，不过大部分都掉在了地上。请重新摆放这些磁贴，得到一个可以正常工作的程序。

如果你在Max OS X或Linux上运行程序，不要忘记加上这一行代码。

```
#! /usr/local/bin/python3

import cgi
import os
import time
import sys
import yate

print(yate.start_response('text/plain'))
addr = os.environ['REMOTE_ADDR']
host = os.environ['REMOTE_HOST']
method = os.environ['REQUEST_METHOD']
cur_time = time.asctime(time.localtime())
print(host + ", " + addr + ", " + cur_time + ": " + method + ": ",
            end='', file=sys.stderr)
```

现在还没有响应……所以只需把纯文本发回给等待的Web浏览器。

这里没有任何新的内容。

```
print('OK.')
```

```
form = cgi.FieldStorage()
```

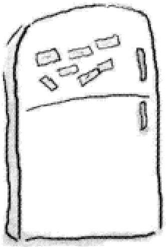
```
end=' ',
```

```
print(each_form_item + '->' + form[each_form_item].value,
```

```
print(file=sys.stderr)
```

```
for each_form_item in form.keys():
```

```
file=sys.stderr)
```



CGI磁贴答案

你需要一个新的CGI脚本，名为add_timing_data.py，它要处理来自表单的数据，并把这些数据显示在Web服务器的控制台屏幕上。这个CGI需要查询环境，将所记录的数据显示在一行上。所有代码都已经有了，不过大部分都掉在了地上。请重新摆放这些磁贴，得到一个可以正常工作的程序。

```
#! /usr/local/bin/python3

import cgi
import os
import time
import sys
import yate

print(yate.start_response('text/plain'))
addr = os.environ['REMOTE_ADDR']
host = os.environ['REMOTE_HOST']
method = os.environ['REQUEST_METHOD']
cur_time = time.asctime(time.localtime())
print(host + ", " + addr + ", " + cur_time + ": " + method + ": ",
            end='', file=sys.stderr)
form = cgi.FieldStorage()
for each_form_item in form.keys():
    print(each_form_item + '->' + form[each_form_item].value,
          end=' ', file=sys.stderr)
print(file=sys.stderr)
print('OK.')
```

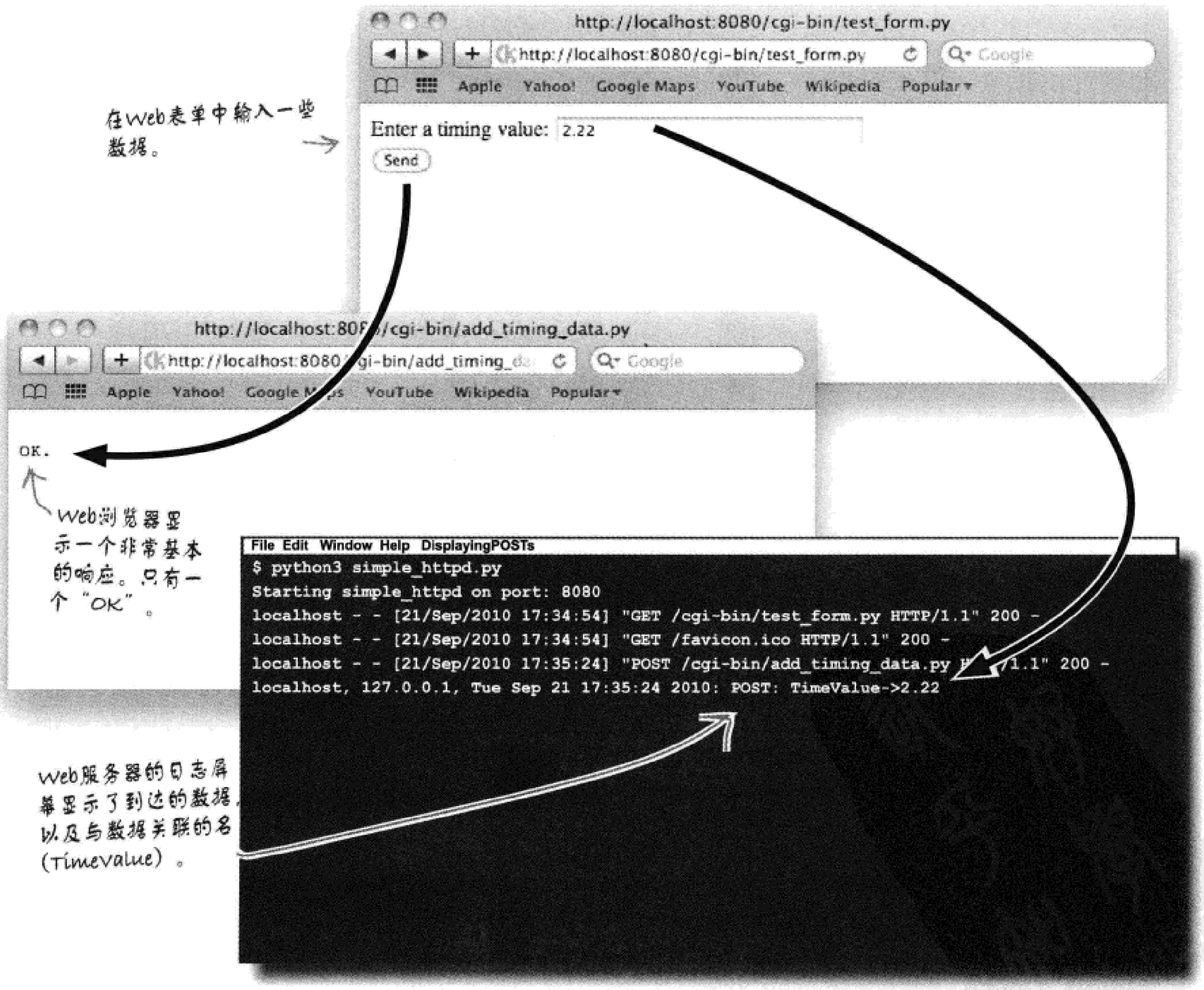
确保这个"print()"函数不会换行。

在标准错误输出上换行。

PDG

测试驱动

下面使用前面的表单生成CGI脚本来尝试add_timing_data.py。在表单中输入数据并按下Send（发送）按钮时，观察Web服务器控制台上会发生什么。



非常完美。输入到表单的数据确实已经传送到服务器上的CGI脚本。你的下一个挑战就是在Android手机上提供同样的用户输入体验。

在Android手机上请求输入

在Android上请求用户输入时，用户看到的对话框与下面这个例子类似，要求用户确认或改变服务器的Web地址和端口。



如果使用`dialogGetInput()`方法，只需一个Android调用就可以创建这个界面：

```
title = 'Which server should I use?'
message = "Please confirm the server address/name to use for your athlete's timing data:"
data = 'http://192.168.1.33:8080'
resp = app.dialogGetInput(title, message, data).result
```

用户与对话框交互的结果赋至"resp"。

按下Ok按钮将把resp设置为输入域中输入的数据。

按下Cancel按钮时，resp设置为None，这是Python的内部null值。

下面创建一些Android数据输入对话框。

Exercise

下面创建一个小Android应用，它与用户交互两次。第一个对话框要求用户确认Web服务器使用的Web地址和端口。假设用户在这个对话框上轻击OK按钮，会弹出第二个对话框，请求用户输入要发送到服务器的计时值。与第一个对话框一样，如果轻击了OK按钮，会继续执行，将新得到的计时值发送到Web服务器。任何情况下只要轻击Cancel按钮就会导致应用退出。

下面已经为你提供了部分代码。你的任务是完成这个程序。在这个代码下面编写你认为需要的代码，将这个程序命名为`get2inputsapp.py`：

这里没有任何新的内容……所有这些代码你之前都见过。

```
import android
from urllib import urlencode
from urllib2 import urlopen

server_title  = 'Which server should I use?'
server_msg    = "Please confirm the server address/name to use for your athlete's timing data:"
timing_title  = 'Enter data'
timing_msg    = 'Provide a new timing value:'
web_server    = 'http://192.168.1.33:8080'
add_time_cgi  = '/cgi-bin/add_timing_data.py'

app = android.Android()

def send_to_server(url, post_data=None):
    if post_data:
        page = urlopen(url, urlencode(post_data))
    else:
        page = urlopen(url)
    return(page.read().decode("utf8"))
```

Exercise Solution

下面创建一个小Android应用，它与用户交互两次。第一个对话框要求用户确认Web服务器使用的Web地址和端口。假设用户在这个对话框上轻击OK按钮，会弹出第二个对话框，请求用户输入要发送到服务器的计时值。与第一个对话框一样，如果轻击了OK按钮，会继续执行，将新得到的计时值发送到Web服务器。任何情况下只要轻击Cancel按钮就会导致应用退出。

下面已经为你提供了部分代码。你的任务是完成这个程序。在这个代码下面编写你认为需要的代码，将这个程序命名为get2inputsapp.py：

```
import android
from urllib import urlencode
from urllib2 import urlopen

server_title  = 'Which server should I use?'
server_msg    = "Please confirm the server address/name to use for your athlete's timing data:"
timing_title  = 'Enter data'
timing_msg    = 'Provide a new timing value:'
web_server    = 'http://192.168.1.33:8080'
add_time_cgi  = '/cgi-bin/add_timing_data.py'

app = android.Android()

def send_to_server(url, post_data=None):
    if post_data:
        page = urlopen(url, urlencode(post_data))
    else:
        page = urlopen(url)
    return(page.read().decode("utf8"))

resp = app.dialogGetInput(server_title, server_msg, web_server).result

if resp is not None:
  web_server = resp
  resp = app.dialogGetInput(timing_title, timing_msg).result
  if resp is not None:
    new_time = resp
    send_to_server(web_server + add_time_cgi, { 'TimingValue' : new_time})
```

第一个对话框请求用户确认要使用的Web地址和端口。

如果用户没有点击Cancel按钮……

……第二个对话框要求输入一个新的计时值。

同样的，如果用户没有轻击Cancel按钮……

……应用将把这个数据发送到Web服务器。

测试驱动

下面使用adb工具把`get2inputsapp.py`复制到模拟器：

```
tools/adb push get2inputsapp.py /sdcard/sl4a/scripts
```

`get2inputsapp.py`应用将出现在SL4A的脚本列表中。继续点击这个应用：

启动这个新应用，现在可以编辑Web服务器地址和端口。

点击输入区时，会弹出Android的"软"键盘。



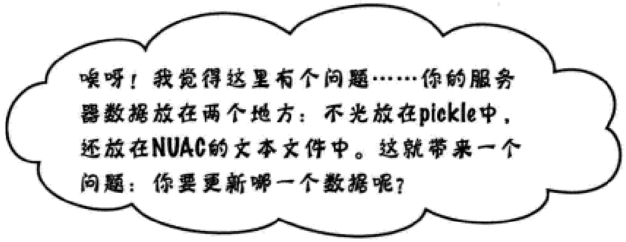
输入一个新的计时值，然后点击"Ok"。

```
File Edit Window Help InputsFromAndroid
$ python3 simple_httpd.py
Starting simple_httpd on port: 8080
localhost - - [21/Sep/2010 17:34:54] "GET /cgi-bin/test_form.py HTTP/1.1" 200 -
localhost - - [21/Sep/2010 17:34:54] "GET /favicon.ico HTTP/1.1" 200 -
localhost - - [21/Sep/2010 17:35:24] "POST /cgi-bin/add_timing_data.py HTTP/1.1" 200 -
localhost, 127.0.0.1, Tue Sep 21 17:35:24 2010: POST: TimeValue->2.22
192.168.1.33 - - [21/Sep/2010 20:50:30] "POST /cgi-bin/add_timing_data.py HTTP/1.1" 200 -
localhost, 192.168.1.33, Tue Sep 21 20:50:30 2010: POST: TimingValue->2:56
```

Web服务器的日志确认了确实由模拟器发送了数据。

非常好，这也能正常工作。不论你的数据来自哪里，不论是Web浏览器还是Android手机，应用都可以把它发送到Web服务器。

该更新服务器数据了

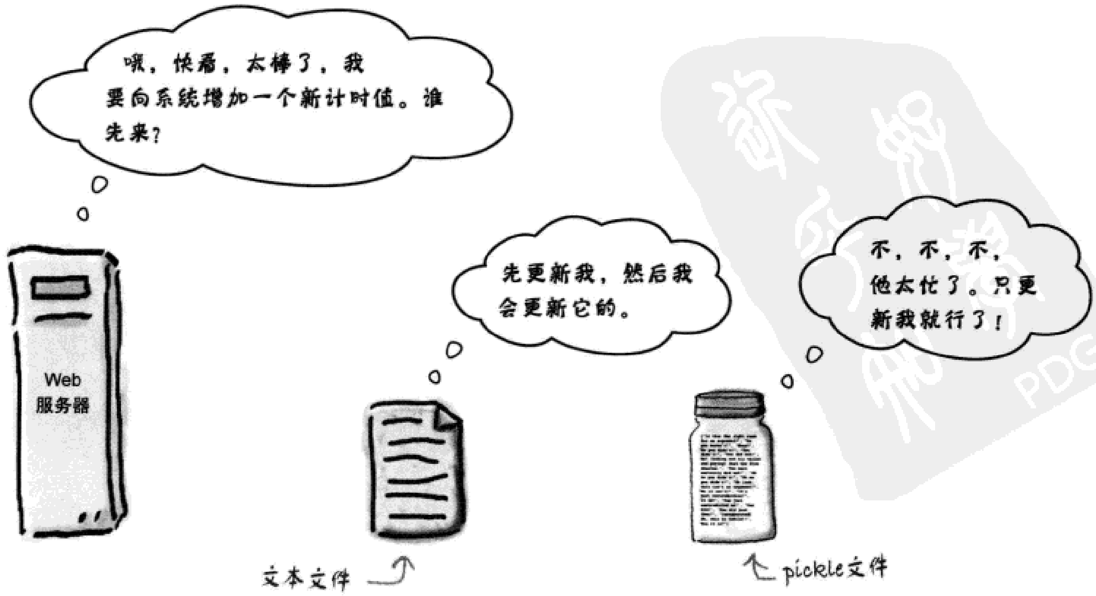




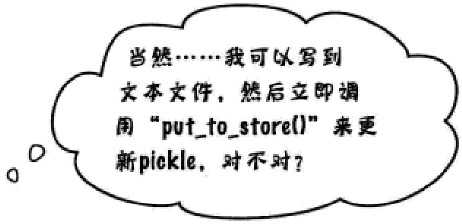
你应该更新哪一个数据集?

如果更新pickle，下一次`put_to_store()`函数运行时，最新的更新就会丢失，因为`put_to_store()`会由文本文件中的数据重新创建pickle。这可不好。

如果更新相应选手的文本文件，pickle中的数据就会过时，直至`put_to_store()`再次运行才能更新。如果在此期间另一个过程调用了`get_from_store()`函数，而此时可能还没有对pickle应用更新，对于读数据的人来说，看起来就好像丢失了更新的数据。这同样也不好。



避免竞态条件





是的，这确实是一种可能的解决方法，但是并不好。

也许你认为，在文本文件更新和pickle重建之间不太可能会有另一个过程调用`get_from_store()`函数……但这确实是可能的，这会导致短时期内数据不一致。这类情况称为竞态条件。一旦发生竞态条件将很难调试。

最好尽你所能不要让这种情况发生。

这里最根本的问题是你对一个数据的更新导致了两个文件交互。即使没有其他不好的影响，起码这也很浪费。





需要一个更好的存储机制

如果只有一个用户在访问数据，只是在这种情况下你原先的文本文件和pickle设计还算不错。不过，现在任何时刻都可能有多个人访问数据，而且可以从任何地方访问。这样一来，你的这种设计就需要改进了。起码要避免这种竞态条件。





there are no Dumb Questions

问： 你肯定应该早就想到这个问题，从一开始就应该“适当”地做出设计，不是吗？

答： 这当然没错，大家都喜欢当“事后诸葛亮”。不过，程序刚开始往往很小，然后会逐步扩展来提供更多的特性，这就会带来复杂性。应该记得教练的这个应用开始时只是一个简单的“独立”应用，它只是一个基于文本的程序，之后转向Web来支持多个用户。再后来又重新开发了部分应用以便在Android手机上使用。当然了，如果我们事先知道所有这些发展，可能就会采用完全不同的设计。

问： 这么说，我必须重写大量代码，是吗？

答： 这要看情况。你使用了MVC模式来构建程序，而且使用了Python，这两点应该能避免重写代码（假设确实需要重写）。


如果我能把数据只放在一个地方，并能支持应用的所有需求，那该多好，不过这可能只是异想天开吧……

使用数据库管理系统

现在需要从结合使用文本文件和pickle的方法转而使用一个真正的数据库管理系统。在这里你有很多选择……



所有这些技术都很棒，也都可行，不过对于你的应用的数据需求来说有些大材小用。而且其中一些远远超出了NUAC的预算，更何况NUAC也没有足够的能力来建立、运行和维护这样一个数据库管理系统。

你需要的系统首先要保证NUAC有能力支持，另外还应当有数据库管理系统所能提供的所有功能。

如果有这样一种技术就好了……

Python包括SQLite

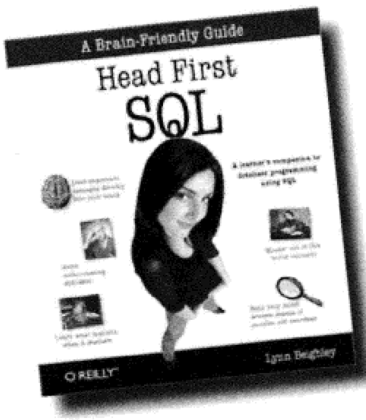
Python 3预装了SQLite 3，这是一个相当完备、无需配置的基于SQL的数据管理系统。

要使用SQLite，只需导入sqlite3库，并使用Python的标准化数据库API来编程。这里确实没有任何其他工作：没有数据库设置，没有配置，也没有以后的维护。

可以把数据存储在SQLite中，重写Web应用的模型代码，使用SQL访问、管理和查询数据。如果应用需求要求迁移到某个更大的数据库系统，也可以做出相应计划。

听起来SQLite非常适合存储NUAC的数据，不是吗？

Geek Bits



这一章中的内容假设你对SQL数据库技术已经很熟悉。如果你还不了解SQL（或者需要简单复习一下），可以看看《Head First SQL》，强烈推荐这本书。

[来自市场的声音：所有出售好书的地方都能找到这本书，只要有信用卡都可以购买。]

利用Python的数据库API

Python数据库API提供了一种标准机制，可以针对各种各样的数据库管理系统编程，其中也包括SQLite。不论使用的后台数据库是什么，代码所遵循的过程都是一样的。

连接
建立与所选数据库后台的一个连接。

创建
创建一个游标，通过连接与数据通信。

交互
利用游标，使用SQL管理数据。

提交
告诉连接对数据应用所做的所有SQL处理，使这些处理永久保留。

回滚
噗！
告诉连接中止SQL处理，将数据返回到交互开始之前的状态。

关闭
撤销与数据库后台的连接。

关闭连接时，游标也会撤销。

数据库API的相应Python代码

下面展示如何使用sqlite3模块实现与数据库的一个交互：

这个磁盘文件用来保存数据库和数据库表。

与以往一样，导入你需要的库。

建立与数据库的连接。

创建数据游标。

执行一些SQL操作。

提交做出的修改，使修改永久保留。

完成时关闭连接。

```
import sqlite3

connection = sqlite3.connect('test.sqlite')

cursor = connection.cursor()

cursor.execute("""SELECT DATE('NOW')""")

connection.commit()

connection.close()
```



取决于这个过程中交互阶段发生的具体情况，对数据的修改可能会永久保留（提交），也可以中止所做的修改（回滚）。

可以在程序中包含类似这样的代码，也可以在IDLE shell中与SQLite数据交互。不论选择哪一种做法，其实都在使用Python与数据库交互。

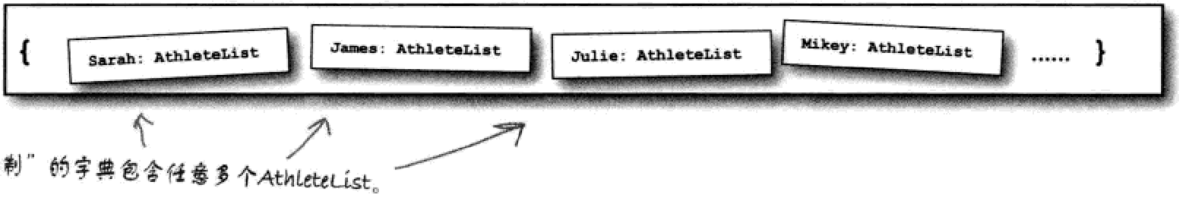
可以使用一个数据库来保存数据，这一点很棒。不过该使用什么模式呢？应当使用一个表，还是需要多个表？要放哪些数据项，另外数据项放在哪里？要如何设计数据库呢？

下面就来回答这些问题。

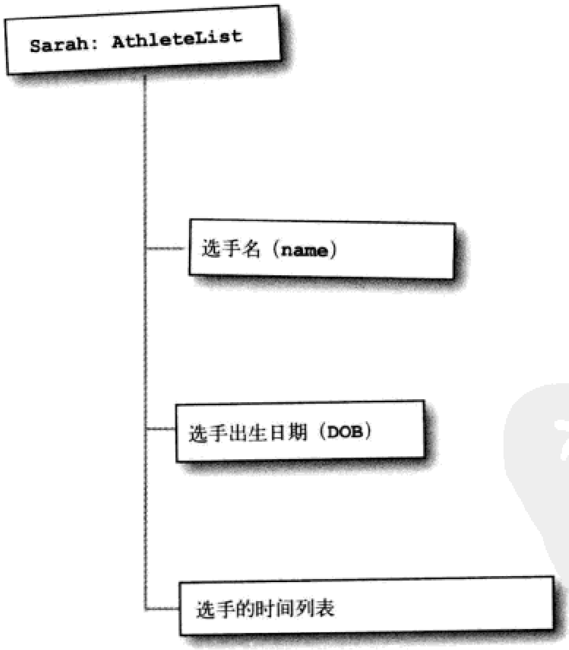
小小的数据库设计会带来很大不同

下面考虑目前NUAC数据在pickle中如何存储。

每个选手的数据是一个`AthleteList`对象实例，与字典中的选手名关联。整个字典“腌制”在pickle中。



每个`AthleteList`有以下属性：



从这个设计可以很清楚地看出哪个名字、出生日期和时间列表与哪个选手关联。不过，如何在一个遵循SQL的数据库系统（如SQLite）中对这些关系建模呢？

需要定义数据库模式并创建一些表。

定义数据库模式

下面是为NUAC数据建议的一个SQL模式。这个数据库名为coachdata.sqlite，它有两个相关的表。

第一个表名为athletes，其中数据行包含一个唯一的ID值，另外还包含选手名和一个出生日期。第二个表名为timing_data，数据行包括选手的一个唯一ID，另外包含具体的时间值。

coachdata.sqlite

这是一个新属性，有了它就能很容易地保证唯一性。

```
CREATE TABLE athletes (
    id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
    name TEXT NOT NULL,
    dob DATE NOT NULL )
```

```
CREATE TABLE timing_data (
    athlete_id INTEGER NOT NULL,
    value TEXT NOT NULL,
    FOREIGN KEY (athlete_id) REFERENCES athletes)
```

注意这个模式使用一个外键将两个表“链接”起来。

在athletes表中对应每个选手有而且只能有一行数据。对于每个选手，ID的值可以保证唯一，这就确保两个（或多个）同名的选手可以在系统中保持独立，因为他们有不同的ID值。

在timing_data表中，每个选手可以有任意多个时间值与其唯一的athlete_id关联，对应所记录的各个时间分别有一行数据。

下面来看一些示例数据。

数据是什么样？

创建这两个表后，从NUAC的文本文件填入数据，表中的数据将与下面的类似。

id	name	dob
1	James Lee	2002-03-14
2	Sarah Sweeney	2002-06-17
3	Vera Vi	2002-12-25
4	Julie Jones	2002-08-17
5	Sally Sanchez	2002-11-24
6	Mikey McManus	2002-02-24

"athletes"表中的数据就是这个样子，对应每个选手只有一个数据行。

"timing_data"表中的数据可能就是这样，对应每个选手可以有多个数据行，每个数据行分别对应一个时间值。

athlete_id	value
1	2.01
1	2.16
1	2.22
1	2.34
1	2.45
1	3.01
1	3.1
1	3.21
2	2.18
2	2.21
2	2.22
2	2.25
2	2.39
2	2.54
2	2.55
2	2.58
3	2.41
3	2.49
3	3.01
3	3.02
3	3.11
3	3.23
4	2.11
4	2.23

表中还包含更多数据，这里显示的只是其中一部分。

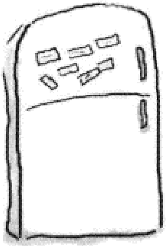
创建这两个表，然后在表中插入数据，NUAC的数据将采用一种更易于处理的格式。

查看这些表，很容易看出如何为选手增加一个新的计时值。只需再向timing_data表增加另外一个数据行。

需要增加一个选手吗？可以向athletes表增加一个数据行。

想知道最快的时间吗？只需从timing_data表的value列中抽取最小的值，对不对？

下面来创建并填充这些数据库表。



SQLite磁贴

下面创建一个小Python程序，它要创建coachdata.sqlite数据库，其中包含空的athletes和timing_data表。将这个程序命名为createDBtables.py。你需要的代码基本已经准备好，请重新摆放这一页下面的磁贴来完成这个程序。

```
import sqlite3

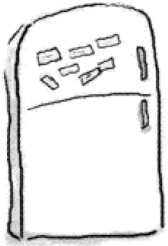
..............................................................

..............................................................
cursor.execute("""CREATE TABLE athletes (
                ................................................
                ................................................
                ................................................

..............................................................
                 athlete_id INTEGER NOT NULL,
                 value TEXT NOT NULL,
                 FOREIGN KEY (athlete_id) REFERENCES athletes)""")

connection.commit()
connection.close()
```

```
cursor = connection.cursor()
dob DATE NOT NULL )""")
cursor.execute("""CREATE TABLE timing_data (
connection = sqlite3.connect('coachdata.sqlite')
name TEXT NOT NULL,
id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
```



SQLite磁贴答案

你的任务是创建一个小Python程序，它要创建coachdata.sqlite数据库，其中包含空的athletes和timing_data表。将这个程序命名为createDBtables.py。你需要的代码基本已经准备好，请重新摆放这一页下面的磁贴来完成这个程序。

```
import sqlite3

connection = sqlite3.connect('coachdata.sqlite')

cursor = connection.cursor()

cursor.execute("""CREATE TABLE athletes (
                    id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
                    name TEXT NOT NULL,
                    dob DATE NOT NULL )""")

cursor.execute("""CREATE TABLE timing_data (
                    athlete_id INTEGER NOT NULL,
                    value TEXT NOT NULL,
                    FOREIGN KEY (athlete_id) REFERENCES athletes)""")

connection.commit()
connection.close()
```

大多数其他数据库系统并不都需要提交，不过SQLite要求提交。

PDG

从pickle向SQLite传输数据

除了编写代码来创建所需要的表，还要将数据从现在的模型（文本文件和pickle组合）传输到数据库模型中。下面编写一些代码来完成这个工作。

可以用SQL INSERT语句为现在的表增加数据。假设变量name和dob中已包含数据，使用如下代码向athletes表增加一个新的数据行：

这些变量中的数据要替换“?”占位符。

```
cursor.execute("INSERT INTO athletes (name, dob) VALUES (?, ?)",(name, dob))
```

不必担心为“id”列提供值，因为SQLite会自动为你提供一个ID值。



现成的Python代码

下面的程序名为initDBathletes.py，它从现有的模型取得选手数据，并将数据加载到新创建的SQLite数据库中。

```
import sqlite3

connection = sqlite3.connect('coachdata.sqlite')
cursor = connection.cursor()

import glob
import athletemodel
data_files = glob.glob("../data/*.txt")
athletes = athletemodel.put_to_store(data_files)

for each_ath in athletes:

    name = athletes[each_ath].name
    dob = athletes[each_ath].dob

    cursor.execute("INSERT INTO athletes (name, dob) VALUES (?, ?)", (name, dob))
    connection.commit()

connection.close()
```

- 连接到新数据库。
- 从现有模型获取数据。
- 从“腌制”数据得到名字和出生日期。
- 使用INSERT语句向athletes表增加新的数据行。
- 使修改永久保留。

为选手指定了什么ID？

需要查询数据库表中的数据来得出为选手自动指定了什么ID值。

对于SQL来说，SELECT语句堪称“查询之王”。下面这个简单的代码段展示了如何在Python中使用SELECT语句，这里假设name和dob变量都有值。

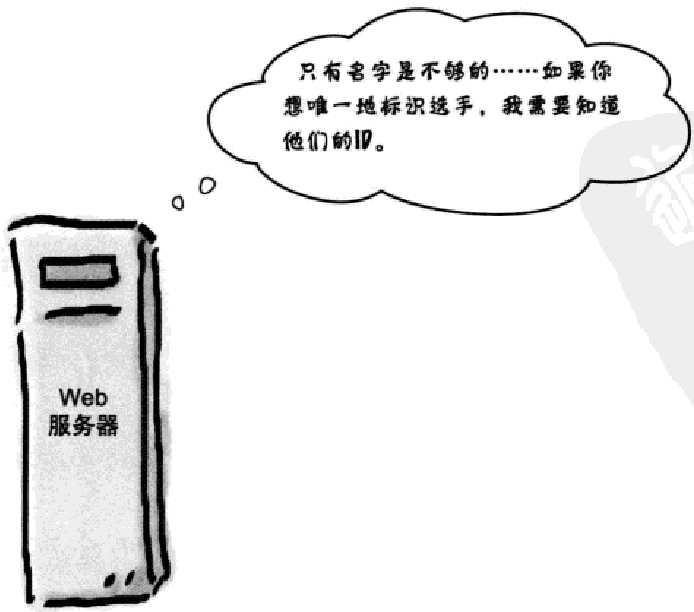
同样的，占位符指示要在哪里把数据值替换到查询中。

```
cursor.execute("SELECT id from athletes WHERE name=? AND dob=?", (name, dob))
```

如果查询成功并返回数据，结果会增加到游标。可以在游标上调用多个不同方法来访问结果：

- cursor.fetchone() 返回下一个数据行。
- cursor.fetchmany() 返回多个数据行。
- cursor.fetchall() 返回所有数据。

这些游标方法分别返回一个数据行列表。



插入计时数据

你已经上路了，下面继续编写代码，从pickle中取出选手的计时值，增加到数据库中。具体来讲，对于pickle中的各个选手，你希望对应与他们关联的各个时间值向`timing_data`表增加一个新的数据行。

Head First代码审查小组那些热心的朋友宣布他们刚刚为你的`AthleteList`类增加了一个`clean_data`属性。访问`clean_data`时，可以得到经过清理和排序的一个计时值列表，而且不会有任何重复。Head First代码审查小组来的真是太及时了。这个属性应该能对你现在的编码工作带来便利。

Sharpen your pencil

拿出笔来，编写所需的代码来查询`athletes`表，得到选手的名字和出生日期，将结果赋至变量`the_current_id`。再写一个查询从pickle中抽取选手的时间，把它们增加到`timing_data`表。

同样的，可以假设你的代码中"name"和"dob"变量已经存在，而且已经赋值。

Sharpen your pencil Solution

拿出笔来，编写所需的代码来查询athletes表，得到选手的名字和出生日期，将结果赋至变量the_current_id。再写一个查询从pickle中抽取选手的时间，把它们增加到timing_data表。

查询athletes表得到ID。

```
cursor.execute( "SELECT id from athletes WHERE name=? AND dob=?",
                (name, dob))
the_current_id = cursor.fetchone()[0]
for each_time in athletes[each_ath].clean_data:
    cursor.execute("INSERT INTO timing_data (athlete_id, value) VALUES (?, ?)",
                   (the_current_id, each_time))
connection.commit()
```

要记住：fetchone()返回一个列表。

通常可以把execute语句分为多行。

取出各个"干净"的时间，并在SQL "INSERT"语句中结合ID使用这个时间。

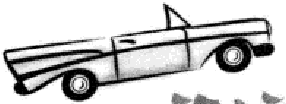
与以往一样，使修改永久保留。

将ID和时间值增加到timing_data表。

动手做！

向initDBathletes.py增加前面的代码，将这个代码增加到connection.commit()调用后面。将程序重命名为initDBtables.py，现在athletes和timing_data表都由一个程序填充数据。

代码已经写得够多了（对现在来说）。下面来传输"腌制"数据。



测试驱动

现在可以运行两个程序：createDBtables.py会创建一个空数据库，并定义两个表，initDBtables.py从pickle抽取数据并填充表。下面不再从IDLE运行这些程序，而是使用Python的命令行工具。

如果你在运行Windows系统，要将“python3”替换为“C:\Python31\python.exe”。

注意这两个程序只能运行一次。

```
File Edit Window Help PopulateTheTables
$ python3 createDBtables.py
$ python3 initDBtables.py
$
```



SQLite数据管理工具

要查看对数据库中数据的管理是否起作用，你有很多选择：

人生苦短。

a **编写更多代码来检查数据库确实是你希望的状态。**
这种方法当然是可行的，不过容易出错，很麻烦，而且要做的工作太多了。

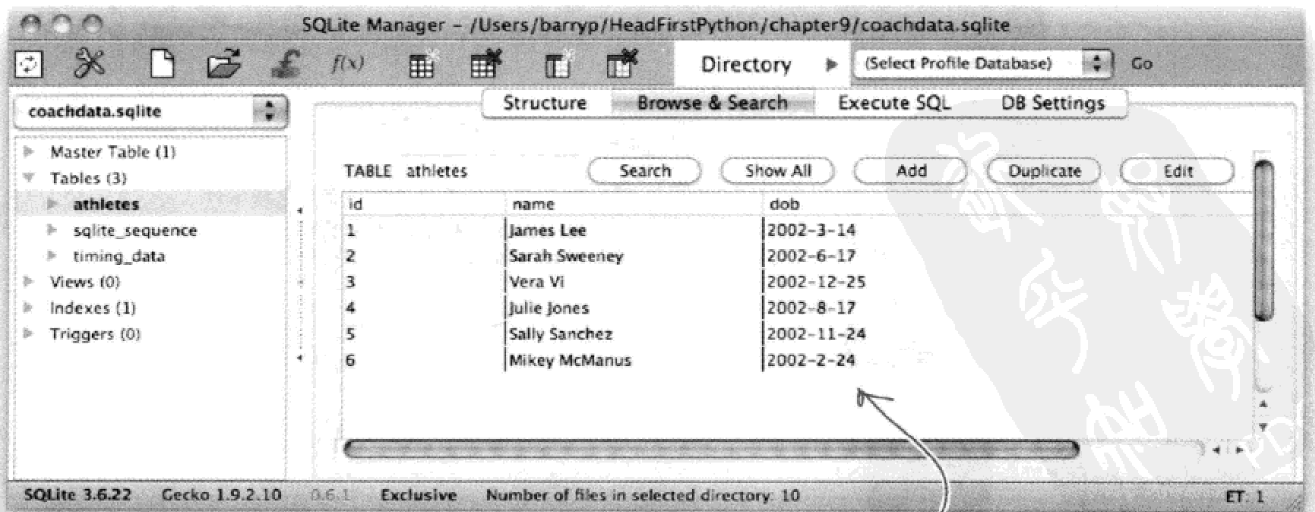
b **使用所提供的"sqlite3"命令行工具。**
只需在终端窗口中键入sqlite3，进入SQLite "shell"。要查看能用哪些命令，可以键入.help，阅读给出的帮助信息。这个工具有些基础（而且有点费解），不过确实是可用的。

这里有一个点号，后面是单词"help"。

c **使用图形化数据库浏览器。**
有很多这样的图形化数据库浏览器，只需在Google中搜索"sqlite database browser"，你会发现大量选择，简直来不及一一查看。我们最喜欢的工具是SQLite Manager，它已经作为一个扩展包安装在Firefox Web浏览器中。

非常棒，但是只能在Firefox上使用。

这就是SQLite Manager。



太棒了，所有选手都在athletes表中。

但是你怎么把这个新数据库集成到你的Web应用中呢？

SQLite与现有Web应用集成



Joe: 这应该很容易。我们只需要重写athletemodel.py中的代码来使用这个数据库，API可以保持不变。

Frank: 你说API保持不变是什么意思？

Joe: 嗯……以get_from_store()函数为例。它返回一个AthleteList字典，所以我们需要确保更新get_from_store()来使用数据库时仍返回一个字典，就像原先一样。

Frank: 哈，现在我懂了：我们可以查询数据库，获取所有数据，把它变成一个包含所有AthleteList对象的大字典，然后把这个字典返回给调用者，是吗？

Joe: 太对了！最好的一点是，调用代码根本不需要任何改变。MVC的这个亮点难道不让你心动吗？

Frank: 嗯……我想是的。

Jim: 咳，咳！

Frank: 怎么了，Jim?

Jim: 你们疯了吗？

Joe & Frank: 怎么了？

Jim: 你们在倒退，只是因为原先是这样设计数据模型的，就只考虑维持与现有API的兼容性。既然已经重新实现模型中存储数据的方式，所以需要考虑是否还需要改变API。

Joe & Frank: 改变API？你才疯了吧？

Jim: 不，我很清醒，只是有些现实。如果可以简化API，重新设计API让它更好地适应我们的数据库，我们就该那么做。

Joe: 那好，不过你要知道，这样一来我们可就该忙了。

Jim: 不用担心,我们的努力肯定是值得的。

Long Exercise

下面花点时间修改模型代码，现在要使用SQLite而不是pickle。首先是 `athletemodel.py` 模块的代码。拿出笔来，划掉不再需要的代码行。

```
import pickle

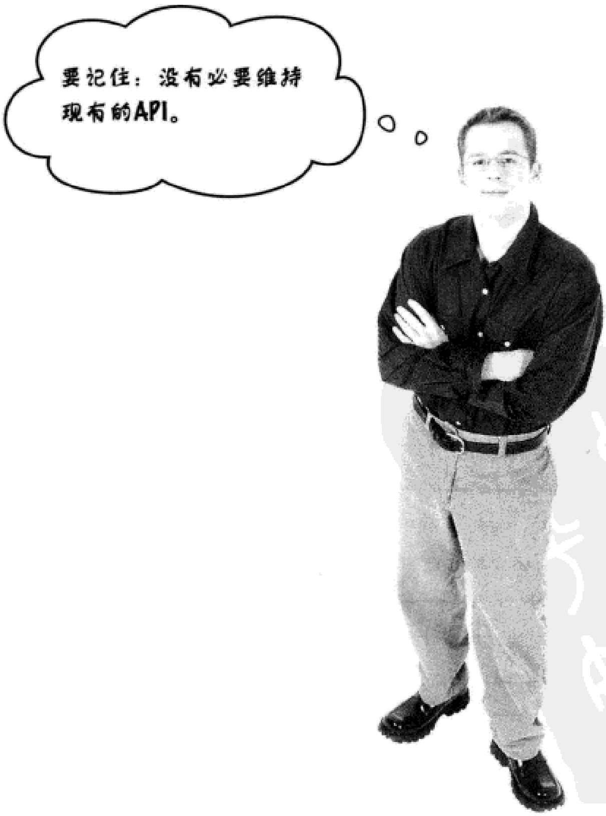

from athletelist import AthleteList


def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return(AthleteList(templ.pop(0), templ.pop(0), templ))
    except IOError as ioerr:
        print('File error (get_coach_data): ' + str(ioerr))
        return(None)


def put_to_store(files_list):
    all_athletes = {}
    for each_file in files_list:
        ath = get_coach_data(each_file)
        all_athletes[ath.name] = ath
    try:
        with open('athletes.pickle', 'wb') as athf:
            pickle.dump(all_athletes, athf)
    except IOError as ioerr:
        print('File error (put_and_store): ' + str(ioerr))
    return(all_athletes)
```

```
def get_from_store():
    all_athletes = {}
    try:
        with open('athletes.pickle', 'rb') as athf:
            all_athletes = pickle.load(athf)
    except IOError as ioerr:
        print('File error (get_from_store): ' + str(ioerr))
    return(all_athletes)

def get_names_from_store():
    athletes = get_from_store()
    response = [athletes[each_ath].name for each_ath in athletes]
    return(response)
```





Long Exercise Solution

下面花点时间修改模型代码，现在要使用SQLite而不是pickle。首先是 `athletemodel.py` 模块的代码。拿出笔来，划掉不再需要的代码行。

```
import pickle

from athletelist import AthleteList

def get_coach_data(filename):
    try:
        with open(filename) as f:
            data = f.readline()
        templ = data.strip().split(',')
        return(AthleteList(templ.pop(0), templ.pop(0), templ))
    except IOError as ioerr:
        print('File error (get_coach_data): ' + str(ioerr))
        return(None)

def put_to_store(files_list):
    all_athletes = {}
    for each_file in files_list:
        ath = get_coach_data(each_file)
        all_athletes[ath.name] = ath
    try:
        with open('athletes.pickle', 'wb') as athf:
            pickle.dump(all_athletes, athf)
    except IOError as ioerr:
        print('File error (put_and_store): ' + str(ioerr))
    return(all_athletes)
```

这些代码都不再需要，因为SQLite会为你提供数据模型。

```
def get_from_store():
    all_athletes = {}
    try:
        with open('athletes.pickle', 'rb') as athf:
            all_athletes = pickle.load(athf)
    except IOError as ioerr:
        print('File error (get_from_store): ' + str(ioerr))
    return(all_athletes)

def get_names_from_store():
    athletes = get_from_store()
    response = [athletes[each_ath].name for each_ath in athletes]
    return(response)
```



仍然需要名字列表

把“原来的”所有模型代码都抛开是可以的，不过仍然需要由模型生成一个名字列表。你决定使用SQLite真是太值了，只需要一个简单的SQL SELECT语句就可以达到目的。



现成的Python代码 下面是新的get_names_from_store()函数的代码：

```
import sqlite3

db_name = 'coachdata.sqlite'

def get_names_from_store():
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()
    results = cursor.execute("""SELECT name FROM athletes""")
    response = [row[0] for row in results.fetchall()]
    connection.close()
    return(response)
```

连接到数据库。

抽取你需要的数据。

建立一个响应。

向调用者返回名字列表。



根据ID得到选手的详细信息

除了名字列表，还要能够根据ID从athletes表抽取选手的详细信息。

现成的Python代码

下面是另一个函数get_athlete_from_id()的代码：

```
def get_athlete_from_id(athlete_id):
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()

    results = cursor.execute("""SELECT name, dob FROM athletes WHERE id=?""",
                                     (athlete_id,))
    (name, dob) = results.fetchone()

    results = cursor.execute("""SELECT value FROM timing_data WHERE athlete_id=?""",
                                     (athlete_id,))
    data = [row[0] for row in results.fetchall()]

    response = {    'Name':   name,
                    'DOB':    dob,
                    'data':   data,
                    'top3':   data[0:3]}
    connection.close()
    return(response)
```

这是一个新函数，获取与指定ID关联的数据。

注意这里使用占位符指示"athlete_id"参数插入到SQL SELECT查询的哪个位置。

从athletes表获取"name"和"DOB"值。

从"timing_data"表得到时间列表。

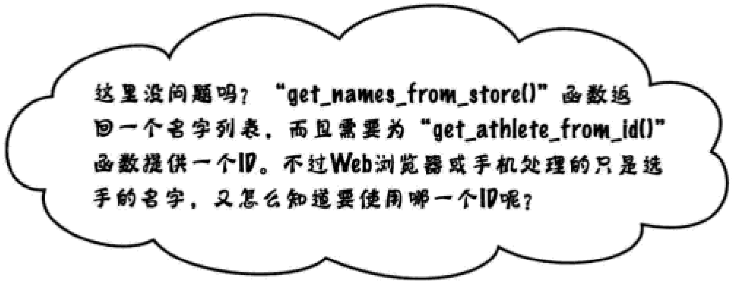
从两个查询结果中取得数据，转换为一个字典。

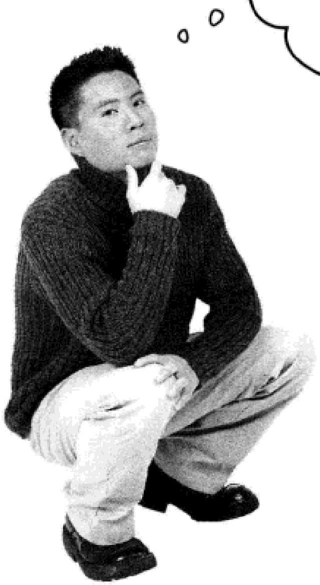
将选手的数据返回给调用者。

这个函数比get_names_from_store()要复杂一些，不过也不算太复杂。这里仍遵循处理SQLite数据时所用的API。一切顺利。

转换模型代码后，现在可以修改CGI脚本来使用这个新的模型API。

下面来看如何修改CGI脚本。





这个问题问得好：要使用哪个ID？

现在的CGI处理的都是选手名，而不是ID。为了确保每个选手是唯一的，所以你将数据库模式设计为包含一个唯一的ID，使系统能够正确地标识同名的两个（或多个）选手，不过目前模型代码并没有向Web浏览器或手机提供ID值。

对于这个问题，一种解决方案是确保在视图中为用户显示选手名，而在系统内部使用ID来唯一地标识一个特定的选手。为此，需要修改`get_names_from_store()`。

Exercise

以下是目前get_names_from_store()函数的代码。不用修改这个代码，可以根据这个代码创建一个新函数，名为get_namesID_from_store()，在响应中不仅包含选手名还要包含ID值。请在下面给出的空白处编写你的新函数。

```
import sqlite3

db_name = 'coachdata.sqlite'

def get_names_from_store():
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()
    results = cursor.execute("""SELECT name FROM athletes""")
    response = [row[0] for row in results.fetchall()]
    connection.close()
    return(response)
```



Exercise Solution

以下是目前get_names_from_store()函数的代码。不用修改这个代码，可以根据这个代码创建一个新函数，名为get_namesID_from_store()，在响应中不仅包含选手名还要包含ID值。请在下面给出的空白处编写你的新函数。

```
import sqlite3

db_name = 'coachdata.sqlite'

def get_names_from_store():
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()
    results = cursor.execute("""SELECT name FROM athletes""")
    response = [row[0] for row in results.fetchall()]
    connection.close()
    return(response)
```

```
def get_namesID_from_store():
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()
    results = cursor.execute("""SELECT name, id FROM athletes""")
    response = results.fetchall()
    connection.close()
    return(response)
```

在SQL "SELECT"查询中包含"id"的值。

并不需要以任何方式处理"results"……只需把查询返回的所有结果赋至"response"。

要记住：关闭连接时，游标也会撤销，所以如果试图使用"return(results.fetchall())"会产生一个异常。

Sharpen your pencil

第1部分：准备好模型代码后，下面修改各个CGI脚本，使它们支持你的新模型。目前，所有代码都假设从模型返回一个选手名列表或AthleteList。拿出笔来，对各个CGI做必要的修改来处理选手ID。

这是"generate_list.py"CGI脚本。

```
#! /usr/local/bin/python3

import glob
import athletemodel
import yate

data_files = glob.glob("data/*.txt")
athletes = athletemodel.put_to_store(data_files)

print(yate.start_response())
print(yate.include_header("NUAC's List of Athletes"))
print(yate.start_form("generate_timing_data.py"))
print(yate.para("Select an athlete from the list to work with:"))
for each_athlete in sorted(athletes):
    print(yate.radio_button("which_athlete", athletes[each_athlete].name))
print(yate.end_form("Select"))
print(yate.include_footer({"Home": "/index.html"}))
```

注意对标题的修改。

这是"generate_timing_data.py"。

```
#! /usr/local/bin/python3

import cgi
import athletemodel
import yate

athletes = athletemodel.get_from_store()
form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value

print(yate.start_response())
print(yate.include_header("NUAC's Timing Data"))
print(yate.header("Athlete: " + athlete_name + ", DOB: " + athletes[athlete_name].dob + "."))
print(yate.para("The top times for this athlete are:"))
print(yate.u_list(athletes[athlete_name].top3))
print(yate.para("The entire set of timing data is: " + str(athletes[athlete_name].clean_data) +
                        " (duplicates removed)."))
print(yate.include_footer({"Home": "/index.html", "Select another athlete": "generate_list.py"}))
```

另一个标题也要修改。

这个"动动笔"练习将在下一页待续，不过先别偷看！修改完这一页上的代码之前先不要翻开下一页。

Sharpen your pencil

第2部分： 先别放下笔！除了修改支持Web浏览器UI的CGI代码之外，还需要修改为Android应用提供Web应用数据的CGI。请修改下面的CGI。

这是“generate_names.py” CGI。

```
#! /usr/local/bin/python3

import json

import athletemodel
import yate

names = athletemodel.get_names_from_store()

print(yate.start_response('application/json'))
print(json.dumps(sorted(names)))
```

这里是“generate_data.py” CGI。

```
#! /usr/local/bin/python3

import cgi
import json
import sys

import athletemodel
import yate

athletes = athletemodel.get_from_store()

form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value

print(yate.start_response('application/json'))
print(json.dumps(athletes[athlete_name].as_dict))
```

Sharpen your pencil Solution

第1部分：准备好模型代码后，下面修改各个CGI脚本，使它们支持你的新模型。目前，所有代码都假设从模型返回一个选手名列表或AthleteList。拿出笔来，对各个CGI做必要的修改来处理选手ID。

这是"generate_list.py" CGI脚本。

```
#! /usr/local/bin/python3

import glob
import athletemodel
import yate

data_files = glob.glob("data/*.txt")
athletes = athletemodel.put_to_store(data_files)

print(yate.start_response())
print(yate.include_header("NUAC's List of Athletes"))
print(yate.start_form("generate_timing_data.py"))
print(yate.para("Select an athlete from the list to work with:"))
for each_athlete in sorted(athletes):
    print(yate.radio_button('which_athlete', athletes[each_athlete].name))
print(yate.end_form("Select"))
print(yate.include_footer({"Home": "/index.html"}))
```

不再需要"glob"模块，因为"get_nameID_from_store()"会为你完成所有这些工作。

get_namesID_from_store()

现在"athletes"是列表的列表，所以要修改代码来得到你需要的数据。

each_athlete[0], each_athlete[1])

radio_button_id() ?

看起来好像需要一个不同的"radio_button()"函数？

这个"动动笔"练习答案的其余部分见下一页。

这是"generate_timing_data.py"。

```
#! /usr/local/bin/python3

import cgi
import athletemodel
import yate

athletes = athletemodel.get_from_store()
form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value
athlete = athletemodel.get_athlete_from_id(athlete_id)
print(yate.start_response())
print(yate.include_header("NUAC's Timing Data"))
print(yate.header("Athlete: " + athlete_name + ", DOB: " + athletes[athlete_name].dob + "."))
print(yate.para("The top times for this athlete are:"))
print(yate.u_list(athletes[athlete_name].top3))
print(yate.para("The entire set of timing data is: " + str(athletes[athlete_name].clean_data) +
                " (duplicates removed)."))
print(yate.include_footer({"Home": "/index.html", "Select another athlete": "generate_list.py"}))
```

从模型得到选手的数据，这会返回一个字典。

根据需要使用返回的数据，访问字典的各个键/值对来得到选手的数据。

athlete['Name'] + ', DOB: " + athlete['DOB']

athlete['top3']

str(athlete['data'])

Sharpen your pencil Solution

第2部分：先别放下笔！除了修改支持Web浏览器UI的CGI代码之外，还需要修改为Android应用提供Web应用数据的CGI。请修改下面的CGI。

这是"generate_names.py" CGI。

```
#! /usr/local/bin/python3

import json

import athletemodel
import yate

names = athletemodel.get_names_from_store()    # 改为：get_namesID_from_store()

print(yate.start_response('application/json'))
print(json.dumps(sorted(names)))
```

这是"generate_data.py" CGI。

```
#! /usr/local/bin/python3

import cgi
import json
import sys

import athletemodel
import yate

athletes = athletemodel.get_from_store()    # 已划掉

form_data = cgi.FieldStorage()
athlete_name = form_data['which_athlete'].value
athlete = athletemodel.get_athlete_from_id(athlete_id)
print(yate.start_response('application/json'))
print(json.dumps(athletes[athlete_name].as_dict))
```

只需对这些CGI稍做修改，因为你的Android应用只关心Web应用的数据，而不需要所有生成的HTML。

将这个代码增加到"yate.py"来支持单选按钮的创建，从而为按钮提供与按钮文本不同的值。

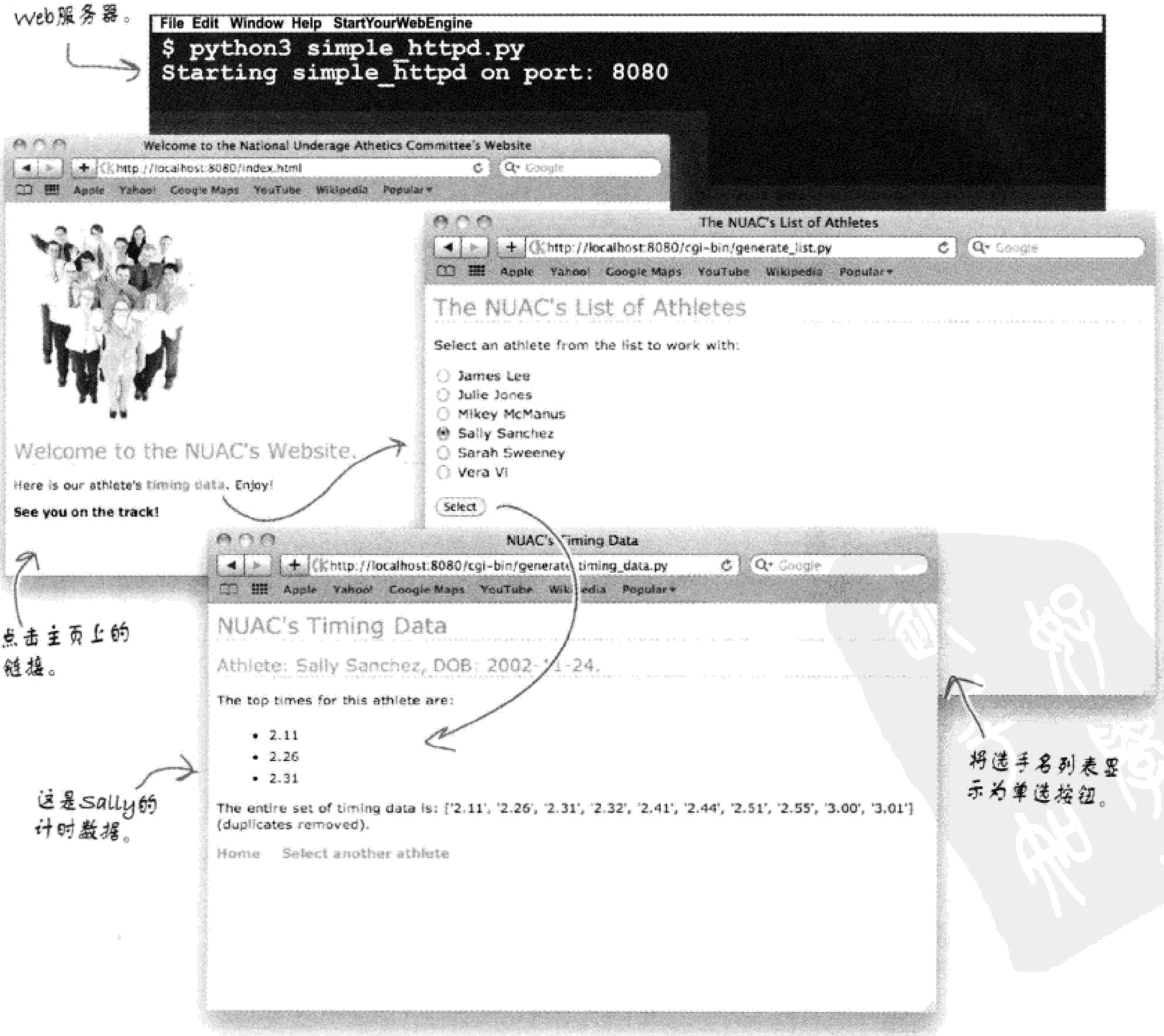
利用第3个参数可以为单选按钮指定一个ID。

```
def radio_button_id(rb_name, rb_value, rb_id):
    return('<input type="radio" name="' + rb_name +
           '" value="' + str(rb_id) + '"> ' + rb_value + '<br />')
```

测试驱动

运行修改后的Web应用之前，一定要把SQLite数据库移至Web应用的顶层目录中（也就是说，移动到index.html文件所在的文件夹）。这样一来，你的模型代码就可以找到这个数据库，所以现在就把它移到Web应用的根文件夹。准备就绪后，来试着运行这个支持SQL的Web应用。

启动（或重启）你的Web服务器。



点击主页上的链接。

将选手名列表显示为单选按钮。

这是Sally的计时数据。

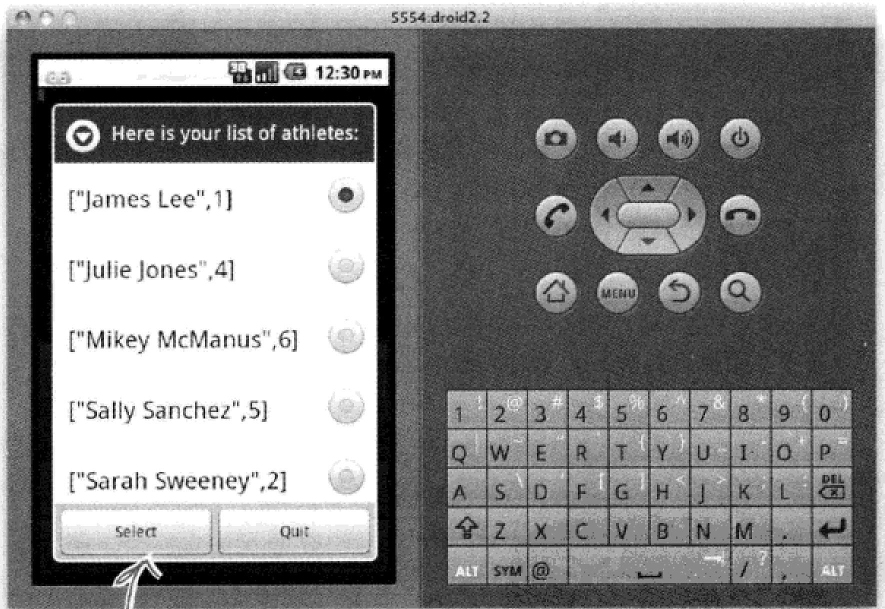
一切正常。不过Android应用呢？

还需要修改Android应用

在Web应用中所有代码都放在Web服务器上，而且在Web服务器上执行，与基于HTML的Web应用不同，Android应用在你的手机上运行，它编写为处理一个名字列表，而不是名字和选手ID的列表。

在模拟器上运行coachapp.py时，会发生奇怪的现象……

这是你目前的Android应用，在模拟器上运行。



这很怪异……应用并不是显示名字，而是显示了一个列表的列表！

如果点击"Select"，应用会崩溃，产生一个"ValueError"。真让人郁闷。

类似于CGI脚本，需要修改Android应用来处理来自Web服务器的数据。也就是说，现在得到的是一个列表的列表，而不是一个名字列表。

完成这个修改并不需要花费太长时间，对不对？

Exercise

以下是目前的coachapp.py代码，需要修改这个代码来支持Web应用模型现在的工作。拿出笔来，对这个代码做必要的修改。

```
import android, json, time

from urllib import urlencode
from urllib2 import urlopen

hello_msg     = "Welcome to NUAC's Timing App"
list_title    = 'Here is your list of athletes:'
quit_msg      = "Quitting NUAC's App."
web_server    = 'http://192.168.1.34:8080'
get_names_cgi = 'cgi-bin/generate_names.py'
get_data_cgi  = '/cgi-bin/generate_data.py'

def send_to_server(url, post_data=None):
    # There is no change to this code from the previous chapter.

app = android.Android()

def status_update(msg, how_long=2):
    # There is no change to this code from the previous chapter.

status_update(hello_msg)
athlete_names = sorted(json.loads(send_to_server(web_server + get_names_cgi)))
app.dialogCreateAlert(list_title)
app.dialogSetSingleChoiceItems(athlete_names)
app.dialogSetPositiveButtonText('Select')
app.dialogSetNegativeButtonText('Quit')
app.dialogShow()
resp = app.dialogGetResponse().result

if resp['which'] in ('positive'):
    selected_athlete = app.dialogGetSelectedItems().result[0]
    which_athlete = athlete_names[selected_athlete]
    athlete = json.loads(send_to_server(web_server + get_data_cgi,{'which_athlete': which_athlete}))
    athlete_title = athlete['Name'] + ' (' + athlete['DOB'] + '), top 3 times:'
    app.dialogCreateAlert(athlete_title)
    app.dialogSetItems(athlete['Top3'])
    app.dialogSetPositiveButtonText('OK')
    app.dialogShow()
    resp = app.dialogGetResponse().result
status_update(quit_msg)
```

Exercise Solution

以下是目前的coachapp.py代码，需要修改这个代码来支持Web应用模型现在的工作。拿出笔来，对这个代码做必要的修改。

```
import android, json, time
from urllib import urlencode
from urllib2 import urlopen

hello_msg     = "Welcome to NUAC's Timing App"
list_title    = 'Here is your list of athletes:'
quit_msg      = "Quitting NUAC's App."
web_server    = 'http://192.168.1.34:8080'
get_names_cgi = 'cgi-bin/generate_names.py'
get_data_cgi  = '/cgi-bin/generate_data.py'

def send_to_server(url, post_data=None):
    # There is no change to this code from the previous chapter.

app = android.Android()

def status_update(msg, how_long=2):
    # There is no change to this code from the previous chapter.

status_update(hello_msg)
athlete_names = sorted(json.loads(send_to_server(web_server + get_names_cgi)))
app.dialogCreateAlert(list_title)
app.dialogSetSingleChoiceItems(athlete_names)
app.dialogSetPositiveButtonText('Select')
app.dialogSetNegativeButtonText('Quit')
app.dialogShow()
resp = app.dialogGetResponse().result

if resp['which'] in ('positive'):
    selected_athlete = app.dialogGetSelectedItems().result[0]
    which_athlete = athlete_names[selected_athlete]
    athlete = json.loads(send_to_server(web_server + get_data_cgi,{'which_athlete': which_athlete}))
    athlete_title = athlete['Name'] + ' (' + athlete['DOB'] + '), top 3 times:'
    app.dialogCreateAlert(athlete_title)
    app.dialogSetItems(athlete['Top3'])
    app.dialogSetPositiveButtonText('OK')
    app.dialogShow()
    resp = app.dialogGetResponse().result
status_update(quit_msg)
```

（手写修改：）

`athlete_names` → `athletes =`

`athlete_names = [ath[0] for ath in athletes]`

只从列表的列表中抽取选手名字。

这是对推导的一个很好的应用。

`which_athlete = athletes[selected_athlete][1]`

确定与所选选手关联的ID。

`athlete['top3']`

需要对下一行做一个小小的调整，访问“top3”属性。

Android池塘谜题



你的**任务**是从池塘里取出代码，并把代码放在以下程序的空白处。你的**目标**是适当地编写程序，使应用为用户提供一种机制，可以向服务器增加对应当前所选选手的计时值。对现在来说，要把数据发送到`cgi-bin/add_timing_data.py` CGI脚本。

提示：这里可以用到（本章前面的）`get2inputsapp.py`的代码。

```
app.dialogSetNegativeButtonText('Add Time')

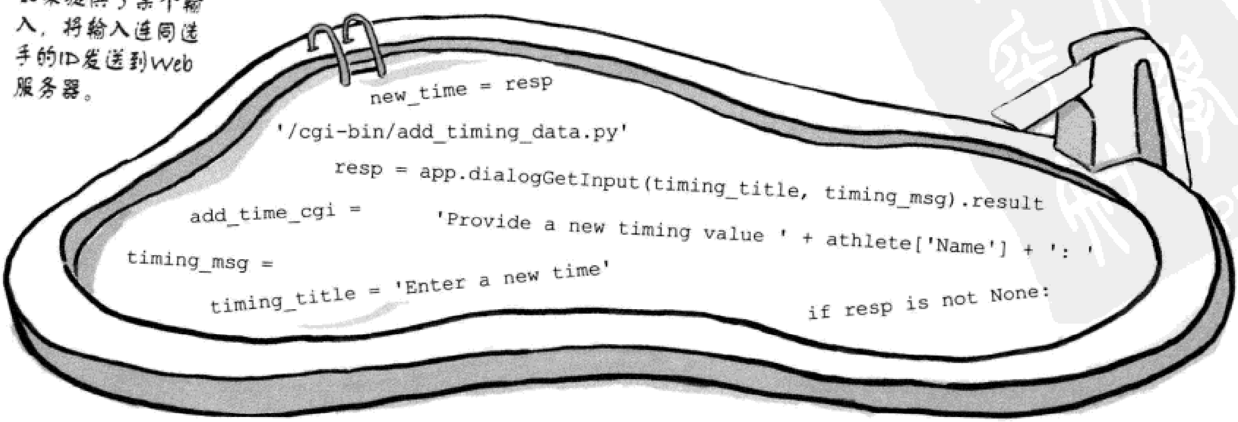
    ...

if resp['which'] in ('positive'):
    pass
elif resp['which'] in ('negative'):
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    ..................................................
    send_to_server(web_server + add_time_cgi,{'Time': new_time, 'Athlete': which_athlete})
```

在当前应用的现有对话框中再增加一个按钮。

根据点击的按钮，可以什么也不做（"pass"），也可以为用户打开一个新对话框。

如果提供了某个输入，将输入连同选手的ID发送到Web服务器。



Android池塘谜题答案



你的**任务**是从池塘里取出代码，并把代码放在以下程序的空白处。你的**目标**是适当地编写程序，使应用为用户提供一种机制，可以向服务器增加对应当前所选选手的计时值。对现在来说，要把数据发送到`cgi-bin/add_timing_data.py` CGI脚本。

提示：这里可以用到（本章前面的）`get2inputsapp.py`的代码。

```
app.dialogSetNegativeButtonText('Add Time')

    ...

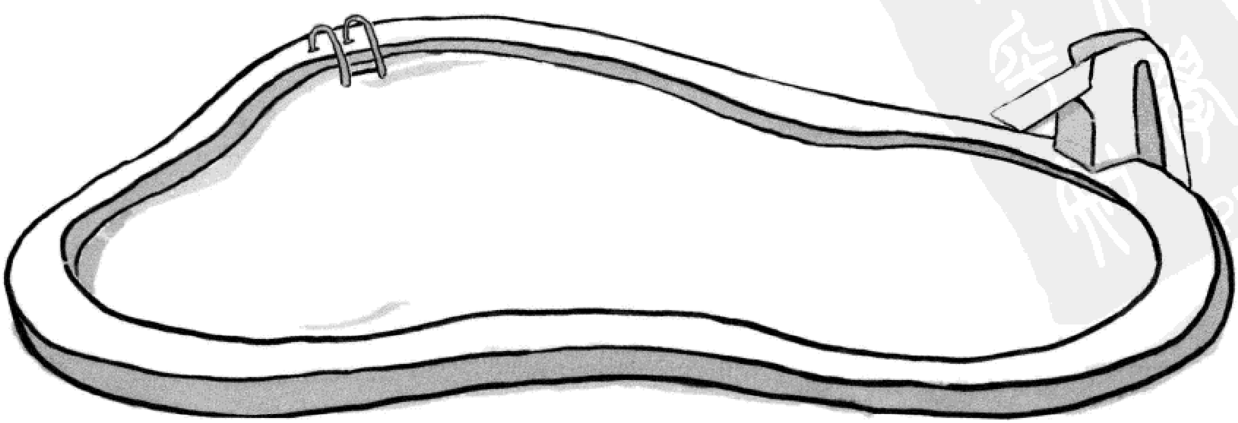
if resp['which'] in ('positive'):
    pass
elif resp['which'] in ('negative'):
    timing_title = 'Enter a new time'
    timing_msg =  'Provide a new timing value ' + athlete['Name'] + ': '
    add_time_cgi ='/cgi-bin/add_timing_data.py'

    resp = app.dialogGetInput(timing_title, timing_msg).result

    if resp is not None:
        new_time = resp
        send_to_server(web_server + add_time_cgi,{'Time': new_time, 'Athlete': which_athlete})
```

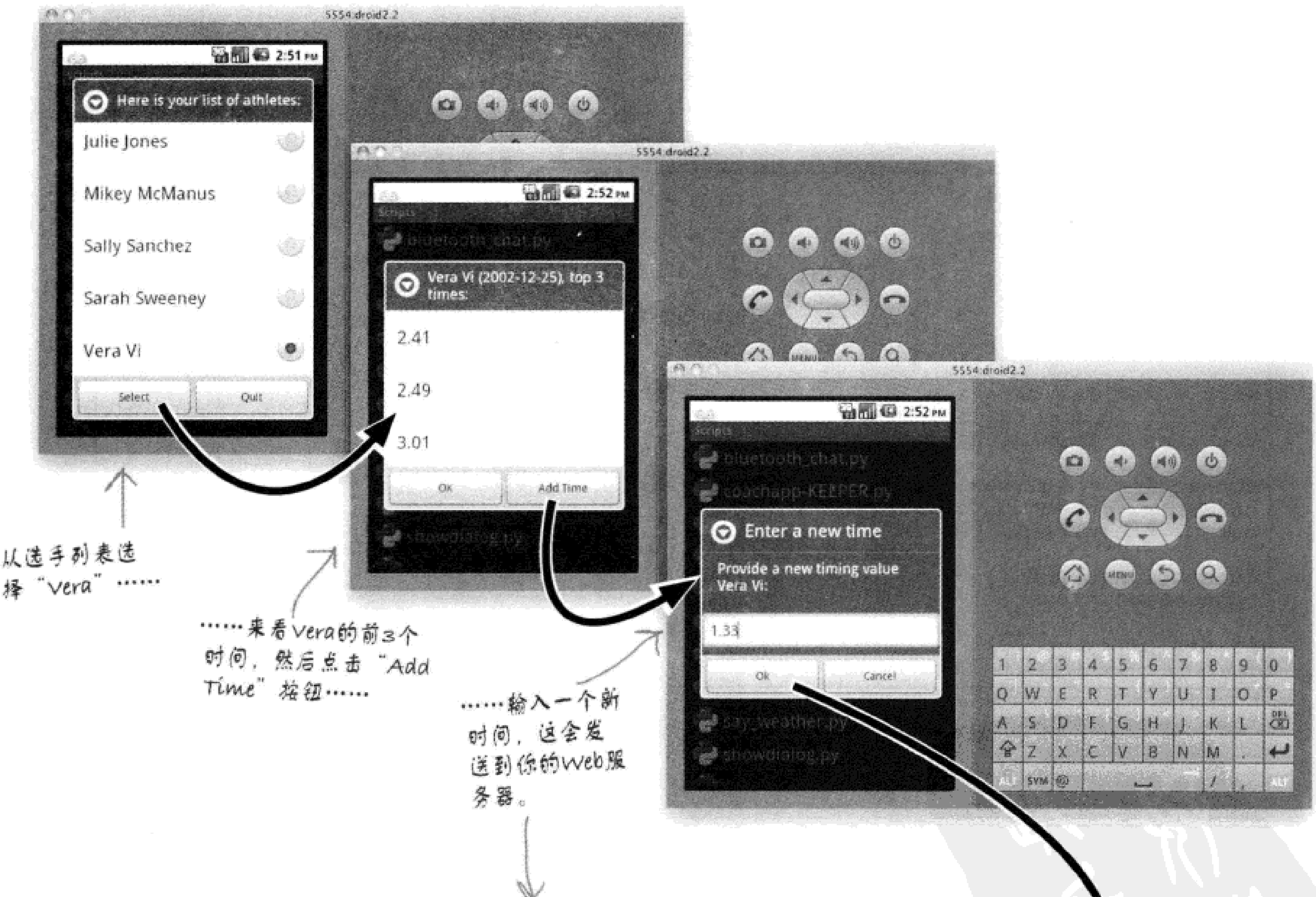
定义对话框的标题，并指定要把数据发送到哪个CGI。

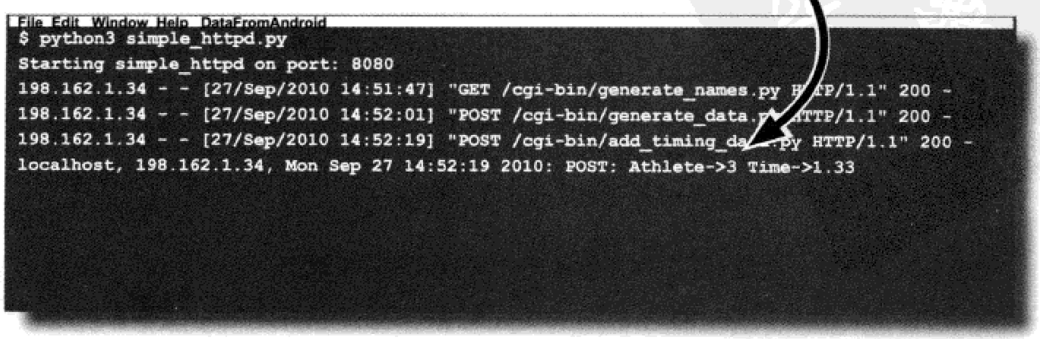
显示对话框并等待用户输入。



测试驱动

使用tools/adb命令把你的最新应用复制到模拟器，然后尝试运行这个应用。





太棒了：你的数据已经从Android应用发送到Web服务器。

更新SQLite中的选手数据

现在只剩下一件事要做，就是修改cgi-bin/add_timing_data.py CGI脚本，将你提交的数据写至数据库，而不是写到Web服务器的控制台屏幕。

对现在来讲，这是一个非常简单的练习，因为只需一个SQL INSERT语句就可以完成所有工作。

add_timing_data.py - /Users/barryp/HeadFirstPython/chapter9/cgi-bin/add_timing_data.py

```
#! /usr/local/bin/python3

import cgi
import sqlite3

import yate

print(yate.start_response('text/plain'))

form = cgi.FieldStorage()
the_id = form_data['Athlete'].value
the_time = form_data['Time'].value

connection = sqlite3.connect('coachdata.sqlite')
cursor = connection.cursor()
cursor.execute("INSERT INTO timing_data (athlete_id, value) VALUES (?, ?)",
                      (the_id, the_time))
connection.commit()
connection.close()

print('OK.')
```

Ln: 22 Col: 0

得到从Android应用发送到Web浏览器的数据。

将数据插入到"timing_data"表。

在Web服务器上运行这个版本的CGI脚本时，任何人在Android手机上输入的任何新时间都会增加到数据库中。

NUAC不必再操心把数据增加到文本文件，因为使用了SQLite，这些文本文件实际上已经过时了。

你已经提供了一个健壮的解决方案，这种方法更可管理、伸缩性更好、更易于编程，而且更易于扩展。这些都要归功于Python的强大功能，归功于它的数据库API以及在标准库中包含了sqlite3。

现在可以坐下来，放松一下，享受你的新成就带给你的荣耀吧……

NUAC非常满意！

当然，使用SQLite并不只是能很容易地插入数据。由于NUAC的数据都放在数据库表中，现在很容易回答他们提出的问题。



要回答针对NUAC数据库中数据的这样一些查询，都要求助于SQL。接下来的工作就可以由你自由发挥了。

你已经将这个Web应用转换为使用一个SQL数据库。随着数据管理需求的增加，还可以考虑采用其他更重量级的数据管理技术。

非常出色。你的Web应用将会大获成功。



你的Python工具箱

你已经读完了第9章，并在你的工具箱里增加了一些重要的Python工具。

Python术语

- "数据库API" —— 这是一种标准化机制，用于从Python程序访问一个基于SQL的数据库系统。

数据库术语

- "数据库" —— 一个或多个表的集合。
- "表" —— 一个或多个数据行的集合，每个数据行包括一个或多个列。
- "SQL" —— 全名是"结构化查询语言"（Structured Query Language），这是数据库世界的语言，允许使用CREATE、INSERT和SELECT等处理数据库中的数据。

BULLET POINTS

- 标准库cgi模块中的`fieldStorage()`方法允许从CGI脚本访问发送至Web服务器的数据。
- 标准os库包含一个`environ`字典，可以很方便地访问程序的环境设置。
- SQLite数据库系统作为`sqlite3`标准库包含在Python中。
- `connect()`方法可以建立与数据库文件的一个连接。
- `cursor()`方法允许通过一个已有的连接与数据库通信。
- `execute()`方法允许通过一个已有的连接向数据库发送一个SQL查询。
- `commit()`方法使之前对数据库所做的修改永久保留。
- `rollback()`方法取消对数据做出的所有未完成的修改。
- `close()`方法关闭与数据库的一个现有连接。
- "`?`"占位符允许在Python代码中为SQL语句指定参数。

10 扩展你的Web应用

来真格的



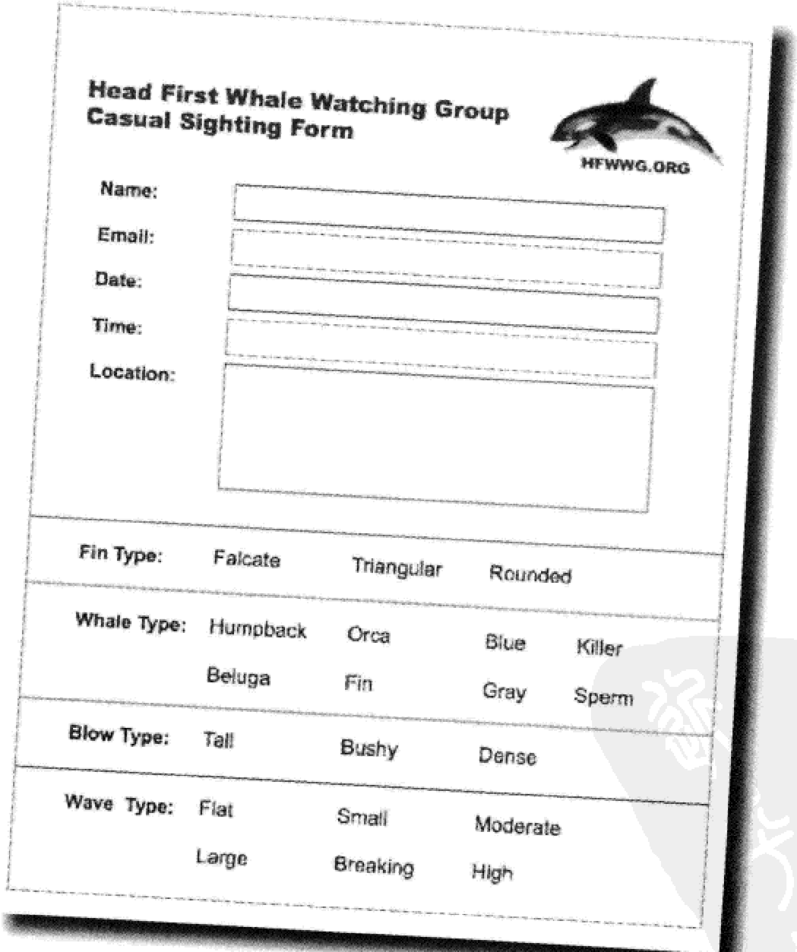
你的应用确实很适合放在Web上……除非开始来真格的。

总有一天你会走运，你的Web应用可能大获成功。等到那一天，你的Web应用不再每天只有寥寥无几的点击量，可能会达到上千、上万，甚至更多。 做好准备了吗？你的Web服务器能处理这么大的负载吗？你又怎么知道它能不能应对？开销有多大？谁来承担费用？你的数据模型能不能扩展到包含数百万的数据项而不会导致应用缓慢如牛？利用Python可以很容易地启动和运行Web应用，而且现在有了Google App Engine，扩展Python Web应用也完全可以轻松实现。不再啰嗦了……请翻开下一页，看看是如何做到的。

到处都有人看到鲸

全国各地如果有人实地观测到鲸，都由Head First观鲸组织（Head First Whale Watching Group，HFWWG）负责协调。目前，他们在网站上提供了一个PDF表格，社会公众可以下载这个表格，填写之后邮寄到HFWWG中心办公室。

这个表格包含记录观鲸所需的基本数据：



Head First Whale Watching Group
Casual Sighting Form

HFWWG.ORG

Name:

Email:

Date:

Time:

Location:

Fin Type:	Falcate	Triangular	Rounded	
Whale Type:	Humpback	Orca	Blue	Killer
	Beluga	Fin	Gray	Sperm
Blow Type:	Tall	Bushy	Dense	
Wave Type:	Flat	Small	Moderate	
	Large	Breaking	High	

这个周末人们频繁地观测到鲸，几千个填好的观鲸表格潮水般向中心涌来……对于数据录入来说这简直是一场噩梦，因为手工地处理所有这些表格可能需要一年时间才能完成。如果你一心想着潜入水中寻找座头鲸，却不得不坐在计算机前面没完没了地录入数据，再没有比这更让人痛苦的了……

HFWWG需要自动化



如果你建议HFWWG投资建设一个成本昂贵的Web托管解决方案，可能不会被采纳。只有周末才会大量出现观鲸记录，如果购买周末才需要的大容量，对于观鲸记录并不多的平时来说完全是个浪费。

如果建议HFWWG投资建一个最前沿的大型Web服务器放在中心办公室，这也很难得到认可。首先没有人能负责这样一个大型设备的管理，另外要处理预想的大流量需要一个宽带链接，而宽带链接的成本太高，会大大超出他们的预算。

还有没有其他的选择?

用Google App Engine构建Web应用

Google App Engine (GAE)是一组Web应用开发技术，允许你在Google的云计算基础设施上托管Web应用。

GAE会持续监视正在运行的Web应用，根据Web应用当前的活动，调整所需的资源来服务Web应用的页面。如果业务很忙，GAE会增加Web应用的可用资源，如果没有太多工作，GAE则会减少资源，直到有另外的活动要求再次增加资源。

在此基础上，GAE还允许访问Google的*BigTable*技术：这是一组数据库技术，利用这种技术，可以非常轻松地存储Web应用的数据。Google还会定期备份Web应用的数据，将Web应用复制到地理位置分散的多个Web服务器上，并保证App Engine每周7天每天24小时都正常运行。

还有最好的一点：可以用Python编写GAE程序。

还有更有诱惑力的吗？你现在就可以免费在GAE上运行你的Web应用。



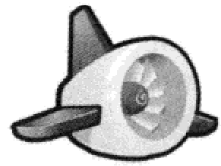
刚开始没有任何障碍。

Google提供了这个Web应用托管服务，而无需支付任何费用，而且会一直提供服务，直到你的Web应用每个月处理的页面访问量达到5000000。一旦超过这个上限，则需要为所使用的额外容量向Google付费。如果你永远也达不到这个限制，那么使用GAE是完全免费的。

5000000页面访问量？这意味着观鲸次数可真不少……

下载和安装App Engine

Web应用准备部署时，可以把它上传到Google云，并从那里运行。不过在开发过程中，可以在你的计算机上本地运行Web应用的一个测试版本。只需要GAE SDK的一个副本，可以从这里得到：



http://code.google.com/appengine/

可以根据你的操作系统下载合适的GAE Python SDK。Windows、Mac OS X和Linux都已得到支持，而且安装很简单。

安装之后，Windows和Mac OS X用户会看到系统上增加了一个漂亮的图形化前端。

在Linux上，成功安装之后会创建一个名为"google_appengine"的新文件夹。

GAE使用Python 2.5

GAE内置的Python版本是Python 2.5的一个修改版本。与使用Python for Android时一样，如果没有运行Python 3，对于GAE来说并不算大问题，不过计算机上确实需要安装有Python 2.5。打开一个终端窗口并键入：

```
python2.5 -V
```

如果这个命令给出一个错误，请到Python网站上针对你的操作系统下载一个2.5版本。

there are no Dumb Questions

问： **是不是在倒退？刚开始是Python 3，然后是Python 2.6 for Android，现在又要为App Engine退步到2.5？怎么回事？**

答： 这个问题问得好。要记住重要的一点，一定要根据给定的限制编写代码。你可能认为GAE在Python 2.5上运行真让人失望，不过事实上并非如此。可以把它认为是对所写代码设置的另一个限制。也就是说，它必须面向Python 2.5版本。与前几章创建的Android代码一样，你要编写的GAE代码与面向版本3的Python代码并没有太大不同。实际上，甚至很难找出二者的差别。

确保App Engine正常工作

Google云中GAE支持的环境支持标准CGI或Python的WSGI。要构建与GAE兼容的Web应用，需要三个东西：一个**文件夹**用来存放Web应用的文件，要执行的**代码**，以及一个**配置文件**。

为了测试GAE的安装，创建一个名为mygaetest的文件夹。在这个文件夹中创建一个可以用来测试GAE的小CGI。将这个CGI命名为`sayhello.py`。使用以下代码：

```
print('Content-type: text/plain\n')

print('Hello from Head First Python on GAE!')
```

再没有比这更简单的了……运行这个CGI，就会在浏览器中显示一个纯文本消息。

配置文件必须名为`app.yaml`，而且这个文件也必须放在Web应用的文件夹中。这个文件告诉Google云有关Web应用运行时环境的一些信息。下面是一个基本的配置文件：

```
application: mygaetest
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
  script: sayhello.py
```

"application"行标识Web应用，它与文件夹同名。

"version"行标识Web应用的当前版本（通常从1开始）。

"runtime"告诉GAE你的Web应用用Python编写并在Python上运行。

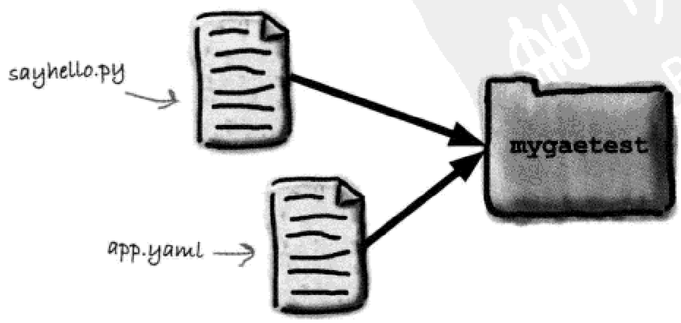
"api_version"指示所面向的GAE版本。

可以把配置文件的"handlers"节认为是一个顶层Web应用路由机制。

这一项告诉GAE将对Web应用的所有请求路由到"sayhello.py"程序。

动手做！

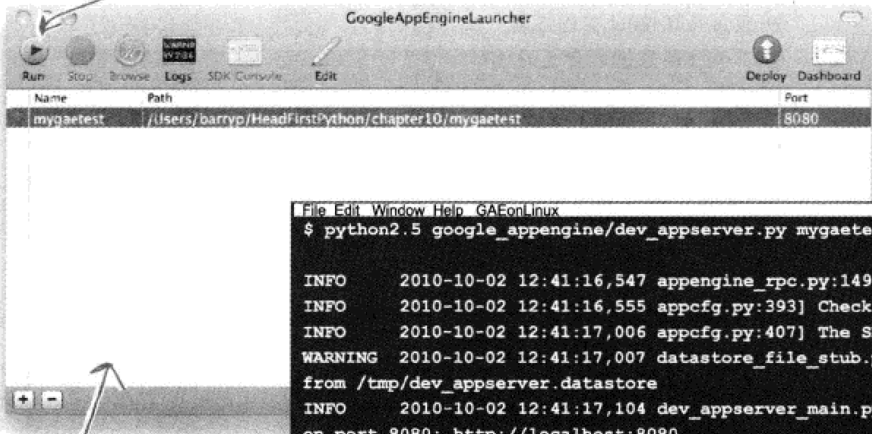
创建一个名为mygaetest的文件夹，并创建这里显示的两个文件。



测试驱动

GAE SDK包含一个测试Web服务器，所以下面使用这个测试Web服务器来尝试运行你的测试GAE Web应用。如果你在Windows或Mac OS X上运行，启动Google App Engine Launcher前端。利用这个工具可以很容易地启动、停止和监视你的Web应用。在Linux上，需要调用一个命令来启动应用。如果你在使用GAE Launcher，可以从菜单系统选择“File” → “Add Existing Application”浏览并选择你的Web应用文件夹。另外：一定要编辑Launcher的首选项，选择Python 2.5作为你的首选Python路径。

点击这个按钮来启动你的Web应用。



这是Mac OS X上的GAE Launcher……看起来与Windows上的界面很类似。

对于Linux，没有图形化前端，所以从命令行启动你的GAE Web应用。

```
File  Edit  Window  Help  GAEonLinux
$ python2.5 google_appengine/dev_appserver.py mygaetest/

INFO     2010-10-02 12:41:16,547 appengine_rpc.py:149] Server: appengine.google.com
INFO     2010-10-02 12:41:16,555 appcfg.py:393] Checking for updates to the SDK.
INFO     2010-10-02 12:41:17,006 appcfg.py:407] The SDK is up to date.
WARNING  2010-10-02 12:41:17,007 datastore_file_stub.py:657] Could not read datastore data
from /tmp/dev_appserver.datastore
INFO     2010-10-02 12:41:17,104 dev_appserver_main.py:431] Running application mygaetest
on port 8080: http://localhost:8080
```

Web应用在端口8080上运行并等待，然后打开你喜欢的Web浏览器，访问Web地址`http://localhost:8080/`。



看吧……这就是来自测试Web应用的消息！



是的，这里确实要做更多工作。不过接下来就会改观了。

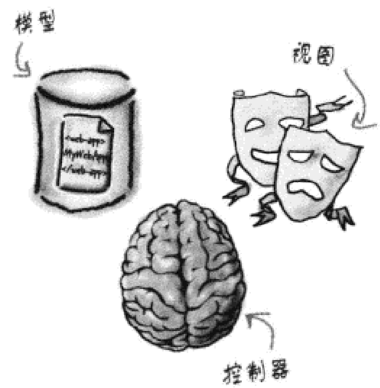
对于现在来说，确实比你原来的工作还要多，但是要记住，这只是一个简单的测试，只是确保你的GAE测试环境能正常运行（而且确实在正常运行）。真正开始处理GAE的一些Web开发特性时，你就会发现除了表面上看到的，在后台还有很多情况。

PDG

App Engine使用MVC模式

Google构建的GAE符合我们熟悉的模型－视图－控制器（MVC）模式。

类似于上一章中的Web应用，支持GAE的Web应用的模型组件使用一个后台数据存储工具，称为datastore（数据存储）。它建立在Google的BigTable技术基础上，这个技术为数据提供了一个“NoSQL”(非SQL)API，另外还使用Google的查询语言（Google's Query Language，GQL）提供了一个类SQL的API。



GAE的视图使用模板，但是不同于上一章中的简单字符串模板，GAE使用了Django项目的**模板系统**，这是Python的一流web框架技术之一。除了模板，GAE还包含Django的表单构建技术。

另外，当然所有控制器**代码**都用Python编写，并且可以使用CGI或WSGI标准。遗憾的是，对GAE不能再使用你的yate模块，因为它是一个Python 3库（要想使用需要大幅重写来支持Python 2）。不用担心：GAE提供的功能足以构建非常棒的Web应用。





没错，与所有其他Web应用的构建过程都一样。

Google努力确保可以尽可能轻松地将Web应用转向App Engine。如果你了解MVC（比如你现在就很了解MVC），就能毫不费劲地基于GAE创建Web应用。只需要知道GAE如何实现各个MVC组件。

用App Engine对数据建模

App Engine将存储在其datastore中的数据项称为属性（properties），这在模型代码中定义。



可以把属性认为是一种定义数据库模式中的数据名和类型的方法：每个属性就像是与行中存储的数据关联的列类型，App Engine将行称为一个实体（entity）。



与传统的基于SQL的数据库一样，GAE datastore属性有一个特定的预声明的类型。有很多类型可供选择，例如：

- db.StringProperty: 一个最多包含500字符的字符串
- db.Blob: 一个字节串（二进制数据）
- db.DateProperty: 一个日期
- db.TimeProperty: 一个时间
- db.IntegerProperty:一个64位整数
- db.UserProperty: 一个Google帐户

所支持的属性类型的完整列表请参考http://code.google.com/appengine/docs/python/datastore/typesandpropertyclasses.html。

这里是上一章的一些示例数据。

这些数据存储为"db.StringProperty"。

这些数据存储为一个"db.IntegerProperty"。

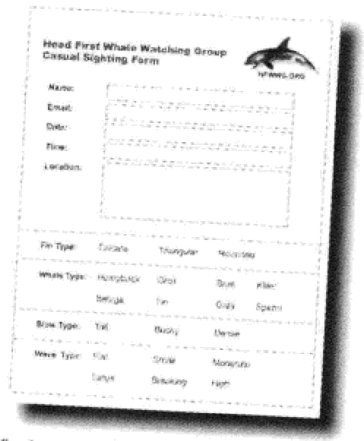
这些数据存储为"db.DateProperty"。

id	name	dob
1	James Lee	2002-03-14
2	Sarah Sweeney	2002-06-17
3	Vera Vi	2002-12-25
4	Julie Jones	2002-08-17
5	Sally Sanchez	2002-11-24
6	Mikey McManus	2002-02-24

池塘谜题



你的任务是从池塘中取出属性，把它们放在类代码中合适的位置，这个代码取自文件hfwwgDB.py。你的目标是为Sighting类中的各个属性分配正确的属性类型。



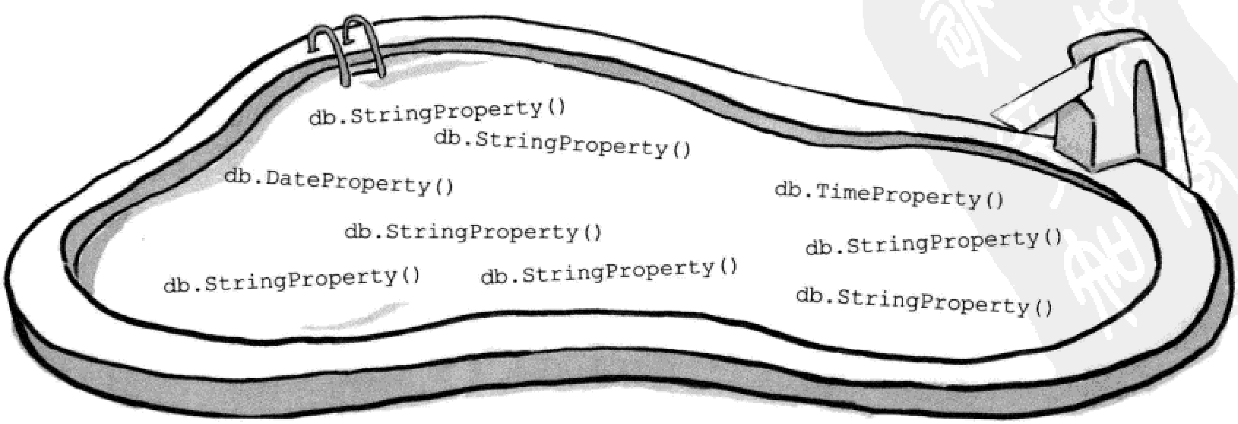
从GAE扩展包导入"db"模块。

```
from google.appengine.ext import db

class Sighting(db.Model):
    name =        ..............................
    email =       ..............................
    date =        ..............................
    time =        ..............................
    location =    ..............................
    fin_type =    ..............................
    whale_type =  ..............................
    blow_type =   ..............................
    wave_type =   ..............................
```

创建一个名为"Sighting"的类，继承自GAE"db.Model"类。

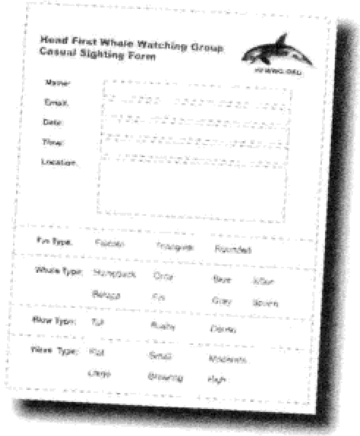
每个属性分别指定一个名。



池塘谜题答案



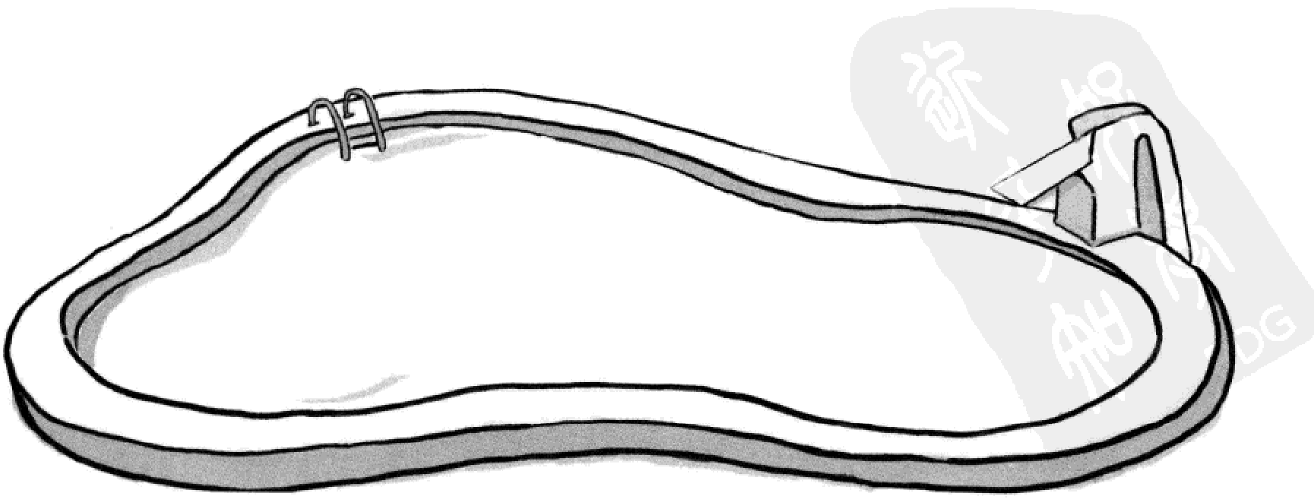
你的任务是从池塘中取出属性，把它们放在类代码中合适的位置，这个代码取自文件hfwwgDB.py。你的目标是为Sighting类中的各个属性分配正确的属性类型。



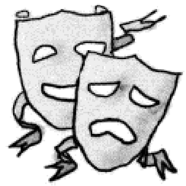

```
from google.appengine.ext import db

class Sighting(db.Model):
    name =          db.StringProperty()
    email =         db.StringProperty()
    date =          db.DateProperty()
    time =          db.TimeProperty()
    location =      db.StringProperty()
    fin_type =      db.StringProperty()
    whale_type =    db.StringProperty()
    blow_type =     db.StringProperty()
    wave_type =     db.StringProperty()
```

除了"date"和"time"字段，所有其他字段都是"StringProperty"。



如果没有视图，模型有什么用？



GAE不仅允许你为数据定义模式，还可以在datastore中创建实体。第一次将数据**放入**datastore时，GAE开始工作，为数据留出空间。你不需要再做额外的工作，只需在代码中定义模型。可以认为GAE会根据需要动态地执行一个类似SQL `CREATE`的命令。不过如何把数据放入GAE datastore呢？

答案很简单：就是由你来放入数据，不过首先需要从Web应用的用户得到一些数据……为此，需要一个视图。使用模板将很容易建立视图。

App Engine模板速览

应该记得，GAE内置的模板技术基于Django项目的技术。Django的模板系统比上一章中基于字符串的简单模板要复杂得多。类似于前面的模板，Django的模板可以将数据替换到HTML中，另外还可以执行条件和循环代码。

下面是HTWWG Web应用需要的4个模板。其中两个你应该已经很熟悉了：它们是对上一章中所用模板的改进。另外两个模板是新增的。可以从本书的支持网站得到这些模板。可以看到，并不是在模板中使用`$name`语句完成变量替换，Django使用了`{{name}}`语法：

header.html

```
<html>
<head>
<title>{{ title }}</title>
</head>
<body>
<h1>{{ title }}</h1>
```

footer.html

```
<p>
{{ links }}
</p>
</body>
</html>
```

form_start.html

```
<form method="POST" action="/">
<table>
```

form_end.html

```
<tr><th> </th><td><input type="submit" value="{{ sub_title }}"></td></tr>
</table>
</form>
```

使用App Engine中的模板

要使用模板，从google.appengine.ext.webapp导入template模块，并调用template.render()函数。可以把template.render()的输出赋至一个变量（在这个代码段中这个变量名为html），这会很有用：

```
from google.appengine.ext.webapp import template

html = template.render('templates/header.html', {'title': 'Report a Possible Sighting'})
```

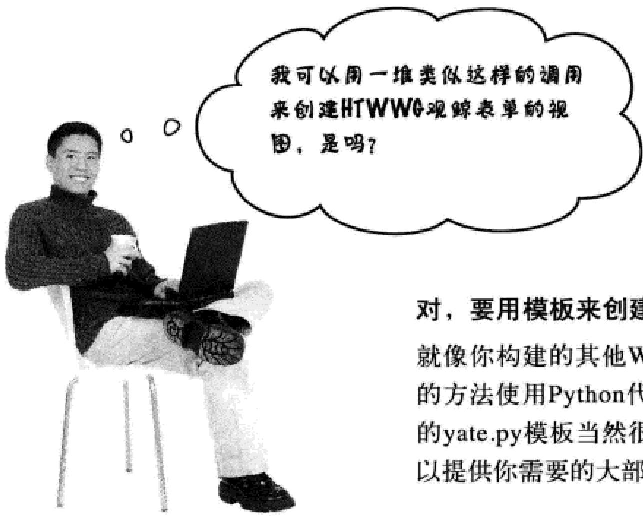
与以往一样，首先是import语句。

调用"template.render()"……

……提供模板名……

……和一个字典，将值映射到指定的模板变量。

这类似于yate.py模板为HTML页面中显示的数据指定参数所用的机制。



对，要用模板来创建你的视图。

就像你构建的其他Web应用一样，可以采用同样的方法使用Python代码来创建视图。不能使用你的yate.py模板当然很遗憾，不过Django的模板可以提供你需要的大部分功能。

there are no Dumb Questions

问： 我要为整个Web页面创建一个大模板吗？

答： 如果你愿意当然可以。不过，如果由模板中的HTML片段来建立视图，就可以在多处重用这些HTML片段。例如，为了维持一种一致的外观，可以在所有Web页面上使用相同的页眉和页脚，当然这里假设页眉和页脚没有嵌入在整个Web页面中（这样就无法重用了）。

Sharpen your pencil

❶ 下面编写创建视图所需的其余代码，这个视图要为HFWWG Web应用显示一个数据输入表单。

除了Web页面页眉代码（已经存在，而且已经提供给你），你还需要编写代码来开始一个新表单，显示表单域，用一个提交按钮结束表单，然后完成整个Web页面。使用所提供的模板，（这里是关键）用不超过4行代码完成这个任务。

这个代码放在一个名为“hfwwg.py”的新程序中。

```
from google.appengine.ext.webapp import template

html = template.render('templates/header.html', {'title': 'Report a Possible Sighting'})

html = html +
```

用所需的其余HTML扩展“html”的内容。

记住，不超过4行代码！

❷ 你尝试用不超过4行代码来编写所需代码，遇到了什么问题？在下面给出的空白处，写下你遇到的问题。

Sharpen your pencil Solution

❶ 下面编写创建视图所需的其余代码，这个视图要为HFWWG Web应用显示一个数据输入表单。

除了Web页面页眉代码（已经存在，而且已经提供给你），你还需要编写代码来开始一个新表单，显示表单域，用一个提交按钮结束表单，然后完成整个Web页面。使用所提供的模板，（这里是关键）用不超过4行代码完成这个任务。

```
from google.appengine.ext.webapp import template

html = template.render('templates/header.html', {'title': 'Report a Possible Sighting'})

html = html + template.render('templates/form_start.html', {})

# 需要在这里生成表单域……不过怎么做呢？

html = html + template.render('templates/form_end.html', {'sub_title': 'Submit Sighting'})

html = html + template.render('templates/footer.html', {'links': ''})
```

"render()"函数总是需要两个参数。如果不需要第二个参数，一定要传入一个空字典。

这是一个问题，对不对？

❷ 你尝试用不超过4行代码来编写所需代码，在下面写下你遇到的问题。

只用4行代码根本不可能做到，因为这样无法生成我需要的表单域。我甚至不能使用"yate.py"的"do_form()"函数，因为那个代码与Python 2.5不兼容……太糟糕了！

你可能会写下这样的文字……当然前提是你没有恼羞成怒，把这本书扔出窗外。☺


如果能避免手写<FORM>，而从一个现有的数据模型生成我需要的HTML标记那该多好，不过我想这可能只是异想天开吧……

Django的表单验证框架

App Engine从Django“借用”的并不只有模板。它还使用了Django的表单生成技术，称为表单验证框架（Form Validation Framework）。给定一个数据模型，GAE可以使用这个框架生成必要的HTML，从而在HTML表中显示表单域。下面是一个示例GAE模型，记录了一个人的基本出生信息：

这个代码放在一个名为“birthDB.py”的文件中。

```
from google.appengine.ext import db

class BirthDetails(db.Model):
    name =          db.StringProperty()
    date_of_birth = db.DateProperty()
    time_of_birth = db.TimeProperty()
```

这个模型与Django的框架结合使用，来生成显示数据输入表单所需的HTML标记。所要做的就是继承GAE包含的一个类，这个类名为`djangoforms.ModelForm`：

```
from google.appengine.ext.webapp import template
from google.appengine.ext.db import djangoforms
import birthDB

class BirthDetailsForm(djangoforms.ModelForm):
    class Meta:
        model = birthDB.BirthDetails
        ...
    html = template.render('templates/header.html', {'title': 'Provide your birth details'})
    html = html + template.render('templates/form_start.html', {})
    html = html + str(BirthDetailsForm(auto_id=False))
    html = html + template.render('templates/form_end.html', {'sub_title': 'Submit Details'})
    html = html + template.render('templates/footer.html', {'links': ''})
```

除了GAE数据模型外，还要导入表单库。

继承“djangoforms.ModelForm”来创建一个新类，然后将这个新类链接到你的数据模型。

用这个新类生成表单。

这里少了一些代码……不过不用担心，稍后就会看到这些代码。对现在来说，只需要重点理解模型、视图代码和Django表单验证框架之间的链接。

检查表单

框架生成了你需要的HTML，并在浏览器中产生以下输出。



这当然不是最漂亮的Web页面，不过确实能用。

使用Web浏览器中的“查看源代码”菜单选项来查看生成的HTML标记。

Source of: http://localhost:8081/

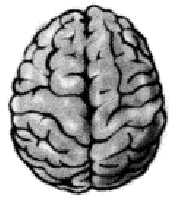
```
<html>
<head>
<title>Provide your birth details</title>
</head>
<body>
<h1>Provide your birth details</h1>
<form method="POST" action="/">
<table>
<tr><th>Name:</th><td><input type="text" name="name" /></td></tr>
<tr><th>Date of birth:</th><td><input type="text" name="date_of_birth" /></td></tr>
<tr><th>Time of birth:</th><td><input type="text" name="time_of_birth" /></td></tr><tr>
<th> </th><td><input type="submit" value="Submit Details"></td></tr>
</table>
</form>
<p></p>
</body>
</html>
```

在代码中将“auto_id”设置为“False”，表单生成器将使用模型属性名来标识表单域。

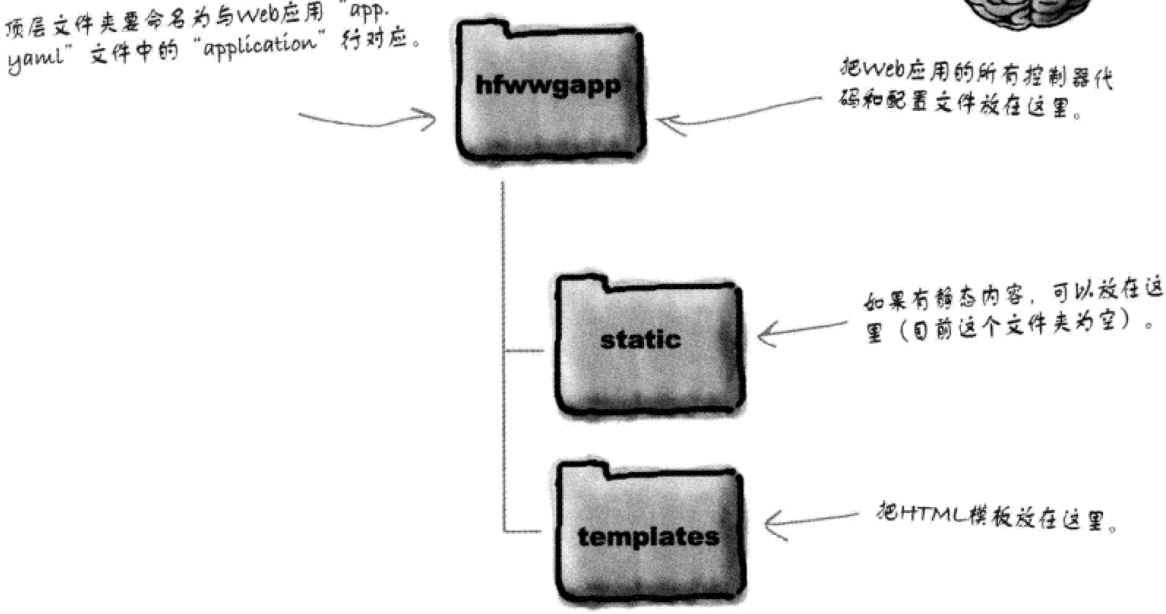
Django框架很聪明，可以为各个输入域创建合适的标签（根据模型中使用的属性名）。

下面用控制器代码把所有组件连接在一起。

控制App Engine Web应用



与所有其他Web应用一样，最好将Web应用控制器代码放在一个特定的文件夹结构中。下面是我们给出的一个建议：



可以看到，任何CGI都可以在GAE上运行，但是为了最大限度地利用Google的技术，编写代码时需要遵循WSGI标准。下面是一些"样板"代码，所有遵循WSGI的GAE Web应用最前面都要有这些代码：

导入一个工具来运行你的Web应用。

导入App Engine的"webapp"类。

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class IndexPage(webapp.RequestHandler):
    def get(self):
        pass

app = webapp.WSGIApplication([('/.*', IndexPage)], debug=True)

def main():
    run_wsgi_app(app)

if __name__ == '__main__':
    main()
```

这个类响应来自Web浏览器的Web请求。

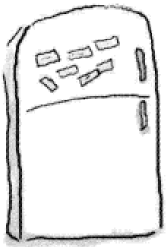
从Web应用接收到一个GET Web请求时会运行这个方法。

为应用创建一个新的"webapp"对象。

这就类似于CGI跟踪的开关。

启动你的Web应用。

只需原样使用这两行代码。



App Engine代码磁贴

下面把所有内容集成在一起。模型代码已经放在hfwwgDB.py文件中。你要做的就是把这个文件移动到Web应用的顶层文件夹中。将模板文件夹也复制到该文件夹下。Web应用的控制器代码放在名为hfwwg.py的文件中，也需要放在顶层文件夹中。现在只有一个问题：有些代码掉到地板上了。请重新摆放这些磁贴来解决这个问题。

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db
from google.appengine.ext.webapp import template
from google.appengine.ext.db import djangoforms
```

所有导入仍然保留……所以没必要重新摆放。

下面考察你是否专心认真。冰箱门上没有给出相关的提示。

这里还缺些什么？

```
        html = template.render('templates/header.html', {'title': 'Report a Possible Sighting'})
        html = html + template.render('templates/form_start.html', {})

        html = html + template.render('templates/form_end.html', {'sub_title': 'Submit Sighting'})
        html = html + template.render('templates/footer.html', {'links': ''})

app = webapp.WSGIApplication([('/.*', SightingInputPage)], debug=True)

def main():
    run_wsgi_app(app)

if __name__ == '__main__':
    main()
```

与样板代码只有一个小的变化，这里并非链接到"IndexPage"。

```
class SightingForm(djangoforms.ModelForm):
```

```
class SightingInputPage(webapp.
RequestHandler):
    def get(self):
```

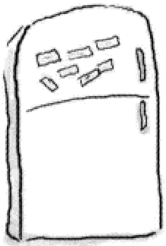
```
html = html + str(SightingForm())
```

```
self.response.out.write(html)
```

```
model = hfwwgDB.Sighting
```

```
class Meta:
```

```
import hfwwgDB
```



App Engine代码磁贴答案

下面把所有内容集成在一起。模型代码已经放在hfwwgDB.py文件中。你要做的就是把这个文件移动到Web应用的顶层文件夹中。将模板文件夹也复制到该文件夹下。Web应用的控制器代码放在名为hfwwg.py的文件中，也需要放在顶层文件夹中。现在只有一个问题：有些代码掉到地板上了。请重新摆放这些磁贴来解决这个问题。

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db
from google.appengine.ext.webapp import template
from google.appengine.ext.db import djangoforms

import hfwwgDB

class SightingForm(djangoforms.ModelForm):
    class Meta:
        model = hfwwgDB.Sighting

class SightingInputPage(webapp.RequestHandler):
    def get(self):
        html = template.render('templates/header.html', {'title': 'Report a Possible Sighting'})
        html = html + template.render('templates/form_start.html', {})
        html = html + str(SightingForm())
        html = html + template.render('templates/form_end.html', {'sub_title': 'Submit Sighting'})
        html = html + template.render('templates/footer.html', {'links': ''})
        self.response.out.write(html)

app = webapp.WSGIApplication([('/.*', SightingInputPage)], debug=True)

def main():
    run_wsgi_app(app)

if __name__ == '__main__':
    main()
```

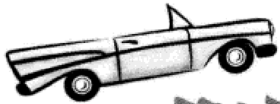
导入GAE数据模型代码。

使用模型来创建一个观察表单，继承自“django.ModelForm”类。

连接的处理器类名为“SightingInputPage”，它提供了一个名为“get”的方法来响应GET Web请求。

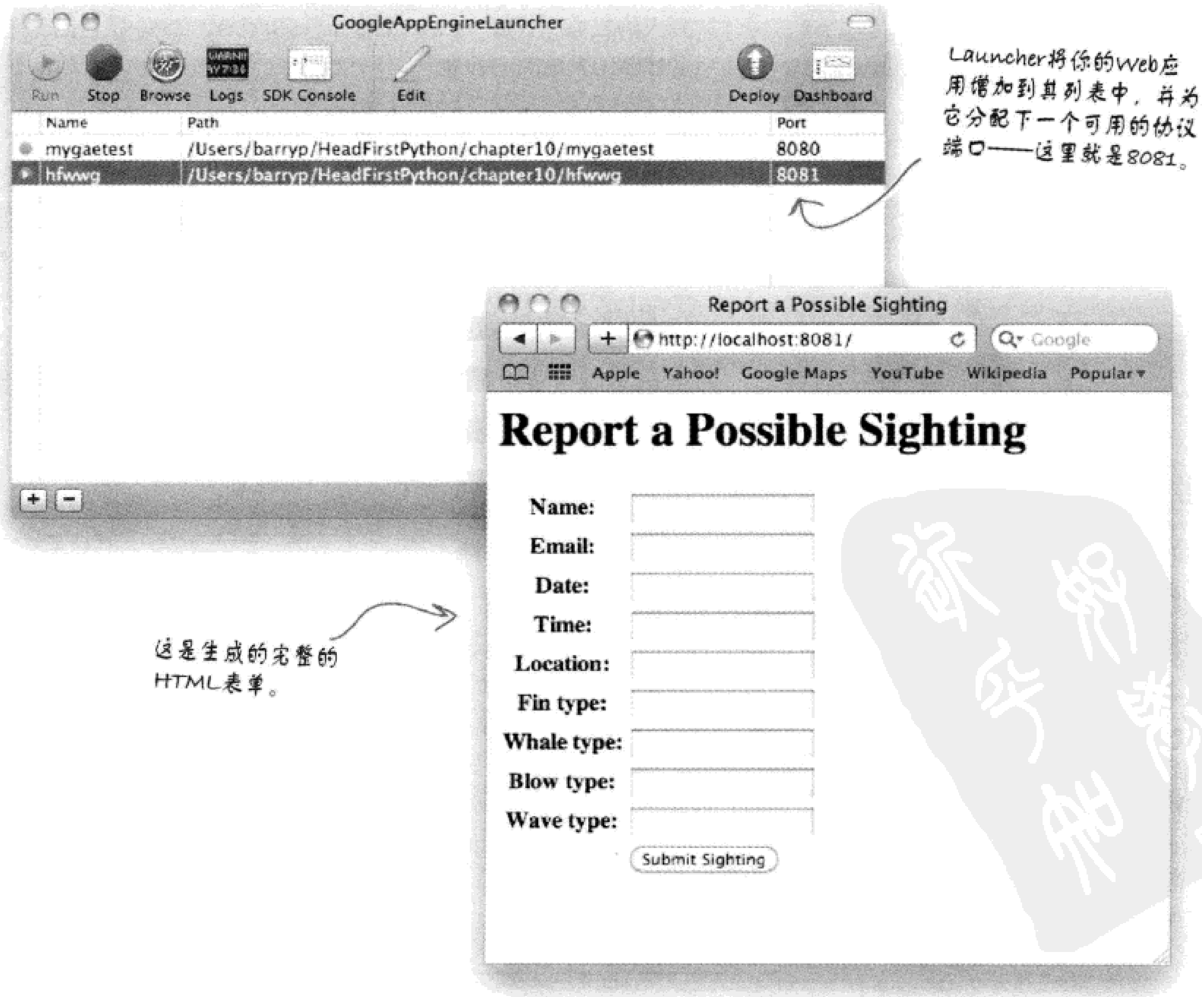
将生成的表单包含在HTML响应中。

这行代码你猜对了吗？需要把一个响应发回等待的Web浏览器，这个工作就由这行代码完成。



测试驱动

真是漫长的过程，不过现在你终于可以测试第一个版本的观鲸表单了。如果你还没有创建app.yaml文件，就请现在着手创建。将application行设置为hfwwg，把script行设置为hfwwg.py。最后一步是使用GAE Launcher中的Add Existing Application（增加现有应用）菜单项，选择你的顶层文件夹作为Web应用的位置。



看起来不错。下面来看HFWWG的人们有何看法。

我知道你肯定在想："穿着这样一件衬衫，这个人怎么可能懂时尚？"……不过我只是想说：你的表单可以稍稍……加点颜色，是不是？这样不是更漂亮一些吗？



OK，我们知道了。不过不用你来操心Web设计。

别担心，你肯定知道代码重用，对吧？所以，下面我们就来重用别人的层叠样式表（Cascading Style Sheets，CSS），来改善生成的HTML表单的"外观"。

不过你能毫无愧疚地向谁"借用"呢？

你很幸运，《Head First HTML with CSS & XHTML》的作者为他们的Web页面创建了大量样式表，你可以直接拿来使用。从这本书的支持网站可以得到他们那些绝妙的样式表（只是稍做了修改）。解压归档文档时，会出现一个名为`static`的文件夹：把这个文件夹整个放在Web应用的顶层文件夹中。

`static`文件夹中有一个文件，名为`favicon.ico`。把它移到你的顶层文件夹中。

改进表单的外观

为了把这些样式表集成到Web应用中，需要为`templates`文件夹中的`header.html`模板增加两个`link`标记。这两个标记应该如下所示：

在"header.html"模板最前面增加这两行代码。

```
<link type="text/css" rel="stylesheet" href="/static/hfwwg.css" />
<link type="text/css" rel="stylesheet" href="/static/styledform.css" />
```

GAE非常聪明，可以优化静态内容的传送，静态内容就是不需要由代码生成的内容。CSS文件都是静态的，放在`static`文件夹中。你要做的就是告诉GAE要启用优化。为此，需要在`app.yaml`文件的`handers`节中增加下面两行代码：

提供静态内容的URL位置。

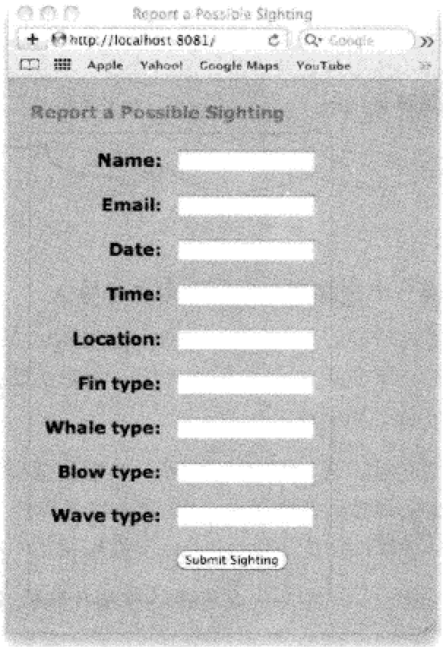
```
- url: /static
  static_dir: static
```

打开优化。

测试驱动

有了样式表，另外修改了app.yaml，现在可以要求浏览器重新加载表单。

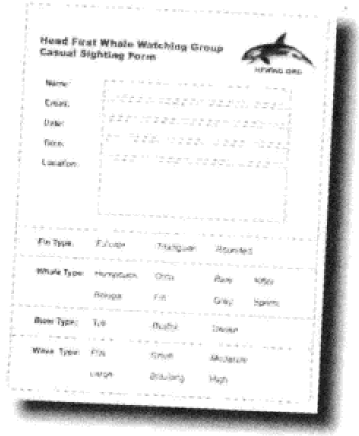
看起来不错。





提供选择来限制输入

目前，表单的Fin、Whale、Blow和Wave输入域中可以接受任何输入。但纸质表格上对于为各个值提供的数据却做出了限制。所以HTML表单也同样应该有限制。





提供一个选择列表来限制用户的输入。

不要对所有表单域都使用HTML的INPUT标记，可以使用SELECT/OPTION标记对来限制表单中某个域可接受的合法数据。这样一来就需要更多HTML标记。这可是个坏消息。

好消息是，表单验证框架可以为你生成需要的HTML标记。只需要在模型代码中定义属性时把要使用的数据项列表作为一个参数（名为choices）。另外还可以对属性使用multiline参数来指示可以接受多行输入。

下面对hfwwgDB.py文件中的模型代码做这些修改。

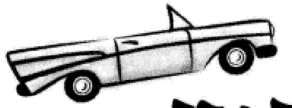
在代码前面定义值列表。

这个命名约定可以帮助标识这些列表包含常量值。

```
_FINS =   ['Falcate', 'Triangular', 'Rounded']
_WHALES = ['Humpback', 'Orca', 'Blue', 'Killer', 'Beluga', 'Fin', 'Gray', 'Sperm']
_BLOWS =  ['Tall', 'Bushy', 'Dense']
_WAVES =  ['Flat', 'Small', 'Moderate', 'Large', 'Breaking', 'High']
                     ......
        location =   db.StringProperty(multiline=True)
        fin_type =   db.StringProperty(choices=_FINS)
        whale_type = db.StringProperty(choices=_WHALES)
        blow_type =  db.StringProperty(choices=_BLOWS)
        wave_type =  db.StringProperty(choices=_WAVES)
```

允许多行输入。

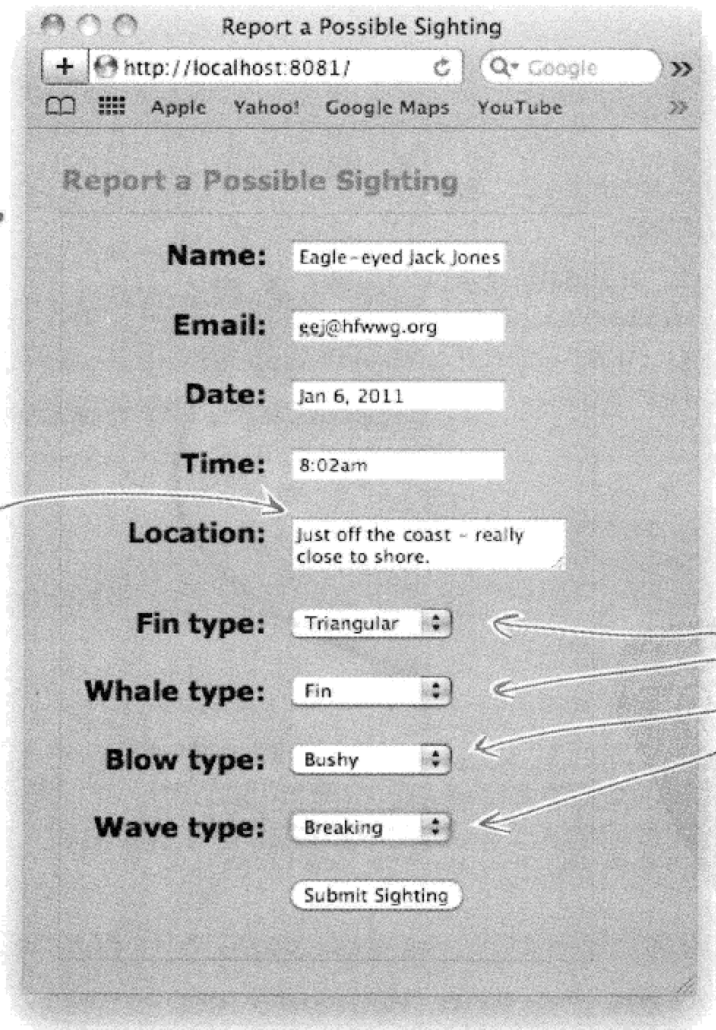
定义属性时使用值列表。



测试驱动

对模型代码应用这些修改后，再次刷新Web浏览器。

表单现在不仅看起来不错，功能也更多了。



"location"域现在是多行显示。

各个"type"域现在分别有一个下拉选择菜单与之关联。

现在表单看起来真不错！输入一些测试数据，然后按下Submit Sighting按钮。

会发生什么？

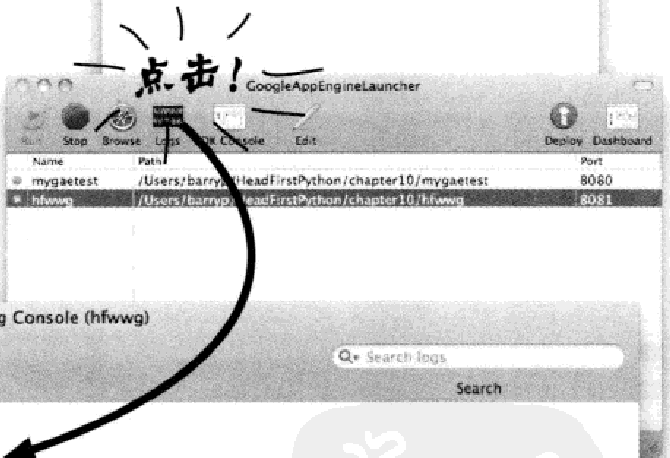
遭遇"死亡白屏"

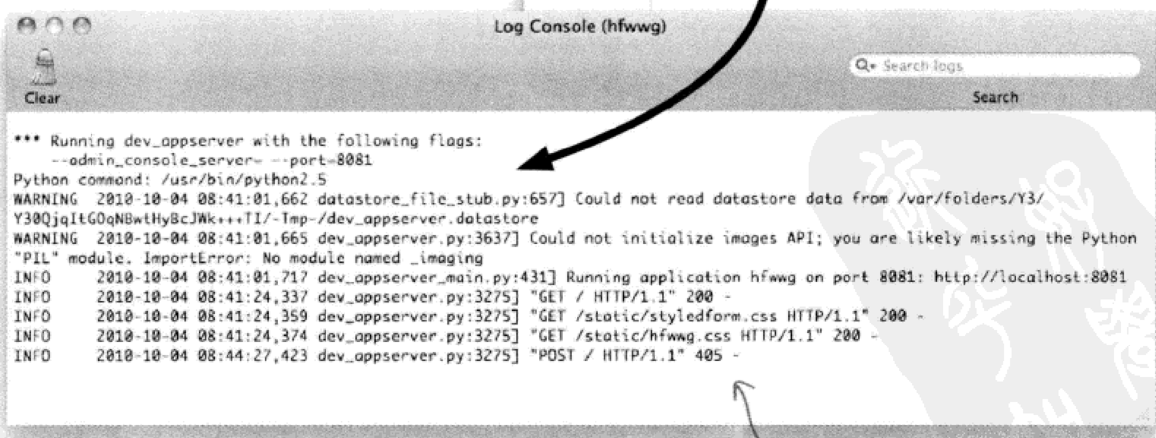
把表单数据提交到GAE Web服务器会出现一个白屏。



要得出发生了什么（或者什么没有发生），需要查看GAE Web应用的日志信息。

如果你在Linux上运行GAE，日志消息会显示在屏幕上。如果在Windows或Mac OS X上运行，可以点击Launcher中的Logs（日志）按钮打开Web应用的Log Console（日志控制台）。





你的请求会导致从web服务器得到一个405状态码。根据官方HTTP RFC标准文档，405表示：

"该方法未得到支持。Request-Line中指定的方法对于Request-URI标识的资源来说不予支持。响应必须包括一个Allow首部，包含所请求资源对应的一个有效方法列表"。

嗯……你是不是反而更糊涂了？

在Web应用中处理POST

405状态码实际上要告诉你：到达Web应用的提交数据没有问题，不过你的Web应用没有办法处理这个请求。这里缺少一个方法。

再来简单查看你的代码：目前只定义了一个方法，名为`get()`。只要有一个GET Web请求到达这个Web应用就会调用这个方法，应该知道，它会显示观鲸表单。

要处理提交的数据，则需要定义另一个方法。具体来讲，需要向`SightingInputPage`类增加一个名为`post()`的新方法。



App Engine既处理请求也处理响应

`get()`方法会生成HTML表单，使用`self.response`对象并在这个对象上调用`out.write()`方法向等待的web浏览器返回一个Web响应。

除了帮你生成Web响应，GAE还可以使用`self.request`对象帮助你处理Web请求。下面几行代码可以显示提交到Web服务器的所有数据：

不要忘记在所有方法上使用“self”。

“arguments()”方法返回表单上使用的表单域名列表。

定义一个名为“post”的新方法。

```
def post(self):
    for field in self.request.arguments():
        self.response.out.write(field)
        self.response.out.write(': ')
        self.response.out.write(self.request.get(field))
        self.response.out.write('<br />')
```

“get()”方法返回与指定表单域名关联的值。

因此……如果你知道表单域的名，就可以在Web应用中使用`self.request.get()`方法来访问它的值。

不过有了数据之后该怎么处理呢？

把数据放在datastore中

数据由GAE发送到Web应用，可以使用self.request.get()方法按域名来访问各个输入域值。应该记得这一章前面的BirthDetails模型：

这个代码放在一个名为“birthDB.py”的文件中。

```
from google.appengine.ext import db

class BirthDetails(db.Model):
    name =          db.StringProperty()
    date_of_birth = db.DateProperty()
    time_of_birth = db.TimeProperty()
```

假设HTML表单已经向Web应用发送数据。这个数据将要存储在GAE datastore中。完成具体工作的代码如下：

```
def post(self):
    new_birth = birthDB.BirthDetails()

    new_birth.name = self.request.get('name')
    new_birth.date = self.request.get('date_of_birth')
    new_birth.time = self.request.get('time_of_birth'))

    new_birth.put()

    html = template.render('templates/header.html', {'title': 'Thank you!'})
    html = html + "<p>Thank you for providing your birth details.</p>"
    html = html + template.render('templates/footer.html',
                      {'links': 'Enter <a href="/">another birth</a>.'})

    self.response.out.write(html)
```

创建一个新的“BirthDetails”对象存放数据。

获取表单的各个数据值，并赋至新对象的属性。

将数据放（保存）在GAE datastore中。

生成一个HTML响应表示“感谢”。

将响应发送到正在等待的Web浏览器。

这里无需多做解释。首先从数据模型**创建**一个新对象，从HTML表单得到数据，将数据赋至对象的属性，然后使用put()方法将数据**保存**到datastore中。

PDG

Exercise

你应该已经知道如何将HTML表单的数据放在GAE datastore中，根据你目前的了解，请创建Web应用现在需要的`post()`方法的代码。我们已经为你完成了部分代码。请提供其余的代码。

```
def post(self):
```

把代码放在这里。

```
    html = template.render('templates/header.html',
                                  {'title': 'Thank you!'})
    html = html + "<p>Thank you for providing your sighting data.</p>"
    html = html + template.render('templates/footer.html',
                 {'links': 'Enter <a href="/">another sighting</a>.'})

    self.response.out.write(html)
```

Exercise Solution

你应该已经知道如何将HTML表单的数据放在GAE datastore中，根据你目前的了解，请创建Web应用现在需要的post()方法的代码。我们已经为你完成了部分代码。请提供其余的代码。

```
def post(self):
    new_sighting = hfwwgDB.Sighting()

    new_sighting.name = self.request.get( 'name' )
    new_sighting.email = self.request.get( 'email' )
    new_sighting.date = self.request.get( 'date' )
    new_sighting.time = self.request.get( 'time' )
    new_sighting.location = self.request.get( 'location' )
    new_sighting.fin_type = self.request.get( 'fin_type' )
    new_sighting.whale_type = self.request.get( 'whale_type' )
    new_sighting.blow_type =self.request.get( 'blow_type' )
    new_sighting.wave_type = self.request.get( 'wave_type' )

    new_sighting.put()

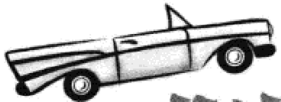
    html = template.render('templates/header.html',
                           {'title': 'Thank you!'})
    html = html + "<p>Thank you for providing your sighting data.</p>"
    html = html + template.render('templates/footer.html',
                 {'links': 'Enter <a href="/">another sighting</a>.'})

    self.response.out.write(html)
```

创建一个新的"Sighting"对象。

对于从HTML表单接收的各个数据值，将它们分别赋至新创建的对象的属性。

将已填充数据的对象存储在GAE datastore中。



测试驱动

将post()代码增加到Web应用中（放在hfwwg.py文件中），在Web浏览器上按“后退”按钮。再次点击Submit Sighting按钮，看看这一次会发生什么情况。

这是你的表单，已经填入数据等待提交。

不过点击这个按钮时，真糟糕……你的Web应用崩溃了。



看起来好像日期属性的格式有点问题，是不是？

唉……真让人扫兴，是吧？

你本来热切盼望表单的数据能存入datastore……但是出现了障碍，没有预想中那么顺利。你认为这是什么问题？

不要破坏“健壮性原则”

健壮性原则（Robustness Principle）指出：“对你发送的内容要保守；对你接收的内容则要宽松自由。”换句话说，从用户请求某种特定类型的数据时不要过于吹毛求疵，但在提供数据时，则要保证用户需要什么就提供什么。

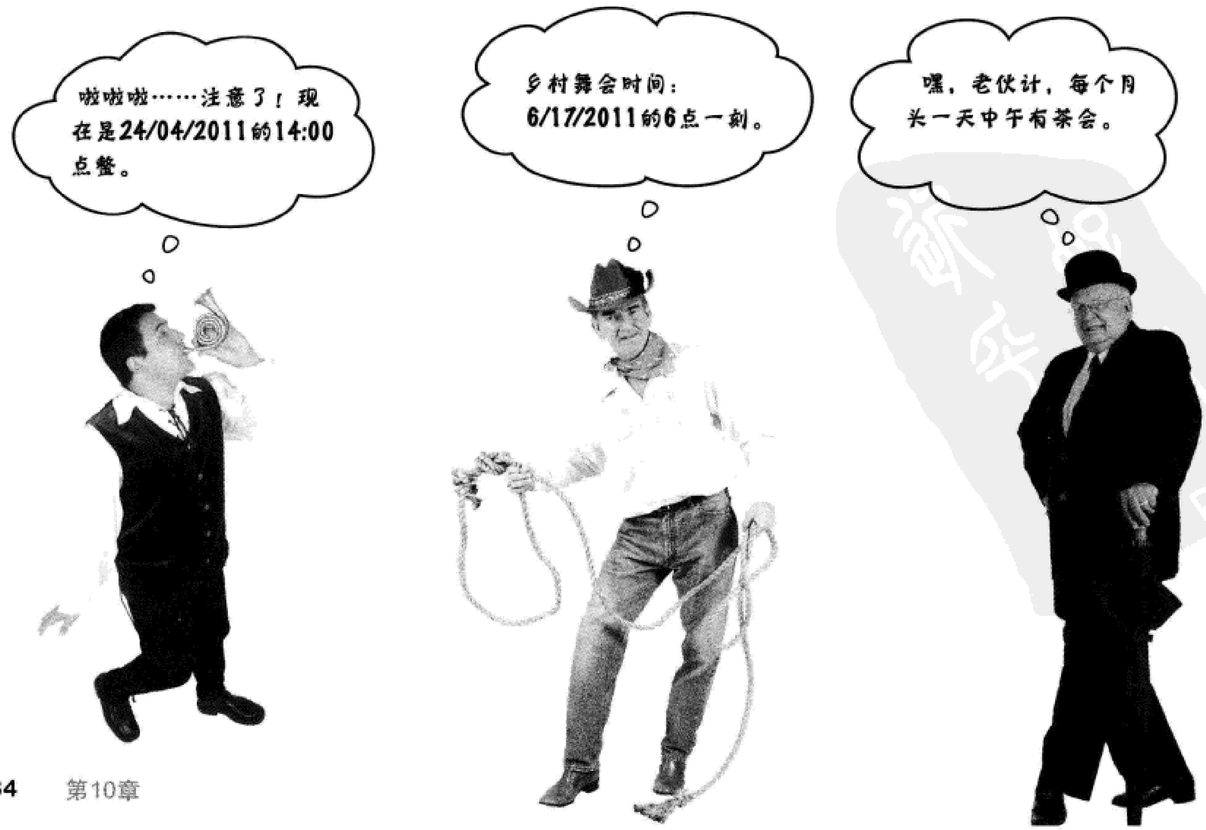
如果用户向系统输入数据时太过困难，情况就不妙了。例如，在你的模型代码中，请考虑日期和时间是如何定义的：

一个日期，只能是日期，而不是其他数据。

```
...
date = db.DateProperty()
time = db.TimeProperty()
...
```

必须为时间提供一个合法的值。所有其他值都不能接受。

现在的麻烦是，指定日期和时间值的方法太多了。



接受几乎所有日期和时间

如果你坚持要求用户提供一个格式正确的日期和时间，就需要在下面的做法中选择其一：

- 指定希望数据采用的格式的详细信息。
- 将输入的数据转换为你可以处理的某种格式。

这两种方法都存在问题。

例如，如果请求日期时过于苛刻，要求必须采用某种特定的格式，这会让你的用户速度慢下来，而且最后可能选择一种对他们完全陌生的日期格式，这会带来困惑。

如果想把输入的所有日期或时间都转换为datastore能理解的一种通用格式，你肯定会受不了的。这会带来很大的复杂性，在这方面可以举一个例子，你怎么知道用户是按mm/dd/yyyy还是按dd/mm/yyyy格式输入数据呢（你是无法知道的）？

还有第三种选择

如果你的应用不需要具体的日期和时间，就不要要求用户输入具体的日期和时间。

当然，其他Web应用对于日期和时间可能没有这么宽松。如果是这样，就需要采用这一页前面讨论的做法之一，尽量满足要求。

对于这个观鲸Web应用，日期和时间可以是无格式的输入域，能够接受任何值（采用任何格式）。重要的是要记录观测到鲸，而不是发现鲸的具体日期/时间。

对日期和时间使用"db.StringProperty()"

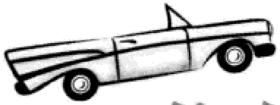
如果放松对date和time域的数据类型限制，不仅能让你的用户更轻松，也能让你自己更轻松。

对于这个观鲸Web应用，解决方案就是在hfwwgDB.py文件中将date和time的属性类型从现在的类型改为db.StringProperty()。

```
...
date = db.StringProperty()
time = db.StringProperty()
...
```

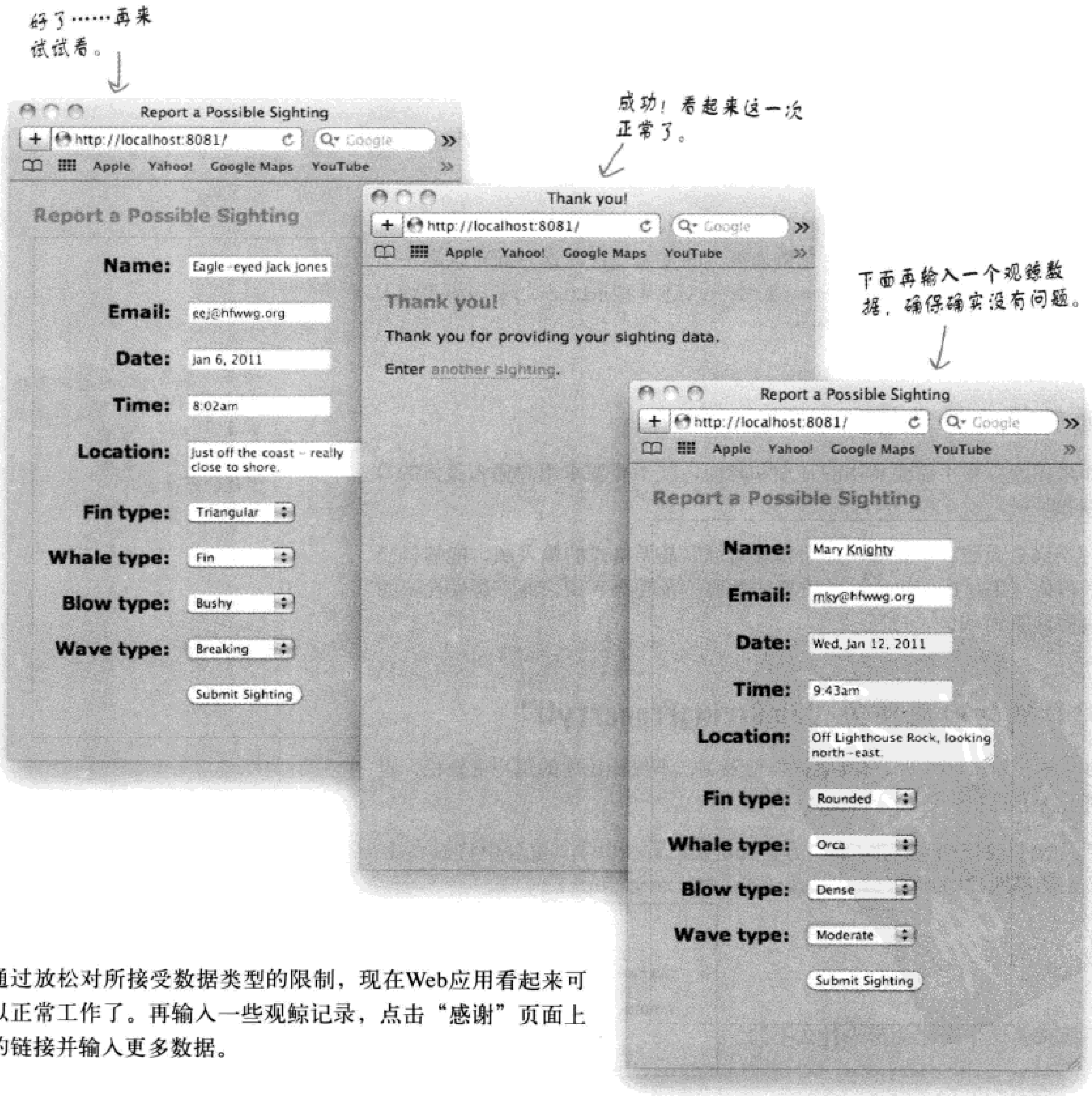
这是一个很小的变化，但是确实会带来很大不同。

下面来看这个变化会带来什么不同。



测试驱动

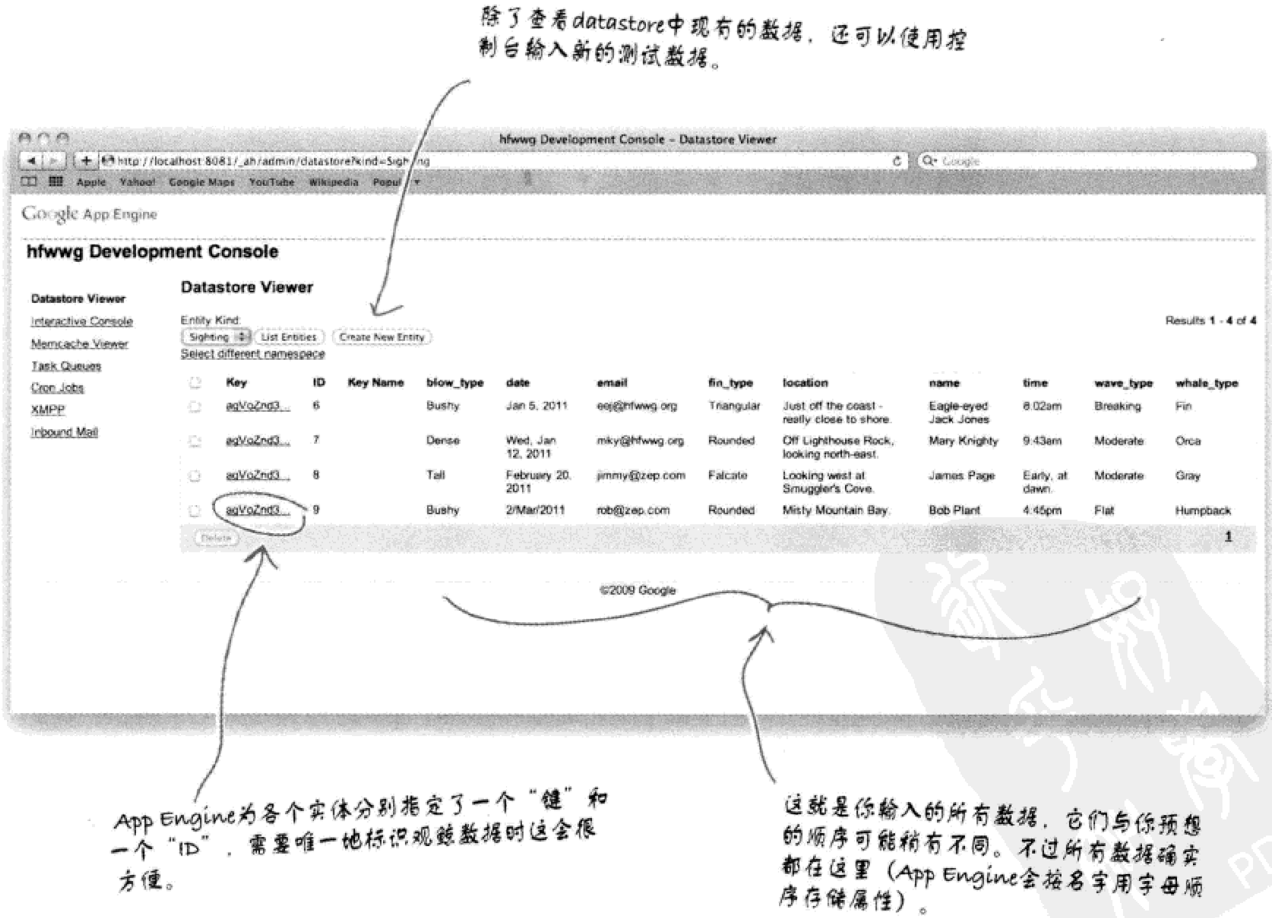
在htwwgDB.py中将date和time的类型改为db.StringProperty()，确保一旦编辑就要保存文件。点击web浏览器中的“后退”按钮，再次提交观鲸数据。



通过放松对所接受数据类型的限制，现在Web应用看起来可以正常工作了。再输入一些观鲸记录，点击“感谢”页面上的链接并输入更多数据。

输入几个观鲸数据后，下面使用App Engine包含的开发控制台来确认这些观鲸数据已经存放到datastore中。

要访问这个控制台，在Web浏览器的地址栏输入`http://localhost:8081/_ah/admin`，并点击List Entities按钮来查看数据。



现在你的GAE Web应用已经日趋完善。

把它部署到Google的云基础设施之前，下面先让HFWWG的人试着运行，看他们是否愿意真正将这个Web应用“投入使用”。

看起来你还没有完成



这会是一个很大的变化吗？

可以想见变化肯定很大。必须创建一个新实体来保存注册用户的登录信息，还需要另外一个表单要求用户提供他们的注册数据（你需要把这些登录数据存储在datastore中）。完成这些工作后，还需要另外一个表单要求用户登录，然后必须提供一种机制来限制仅仅已注册而且登录的用户才能查看Web应用的页面，当然这里假设你可以提供一种切实可行的健壮的机制……

或者……既然这里用到GAE，也可以启用授权。

有时，最小的改变可能会带来天壤之别……

Google的工程师们设计将App Engine部署在Google的云基础设施上。因此，他们决定允许GAE上运行的Web应用访问Google Accounts系统。

通过启用授权（authorization），可以要求Web应用的用户在查看Web应用页面之前先登录到他们的Google帐户。如果用户尝试访问你的Web应用，但是没有登录，GAE会把他重定向到Google Accounts登录和注册页面。成功登录后，GAE会让用户返回到你的Web应用。是不是很酷？

要启用授权，只需对`app.yaml`文件做一个小小的修改：

```
application: hfwwgapp
version: 1
runtime: python
api_version: 1

handlers:
- url: /static
  static_dir: static

- url: /.*
  script: hfwwg.py
  login: required
```

只需做这个小小的修改。

现在，尝试访问你的Web应用时，会要求你先登录。

这就是你的计算机上运行的GAE测试环境中的登录屏幕。



还要捕获用户的Google ID

既然Web应用要求用户登录，下面来得到用户的登录信息，作为观鲸信息的一部分。

首先在hfwwgDB.py文件中为实体的属性列表增加以下属性。把它增加到wave_type属性后面。

```
which_user = db.UserProperty()
```

在你的"sighting"类中创建一个新属性……

……并设置其属性类型。

下面确保Django的表单验证框架在生成HTML表单时不包括这个新属性。在hfwwg.py文件中，将SightingForm类修改如下：

```
class SightingForm(djangoforms.ModelForm):
    class Meta:
        model = hfwwgDB.Sighting
        exclude = ['which_user']
```

确保Django在生成的表单中不包括这个新属性。

仍然在hfwwg.py文件中，在程序前面增加另一个import语句：

```
from google.appengine.api import users
```

导入GAE的Google Accounts API。

在post()方法中，将新的观鲸数据增加到datastore之前，增加下面这行代码：

```
new_sighting.which_user = users.get_current_user()
```

将数据增加到datastore时，这行代码会包括当前登录用户的Google ID。

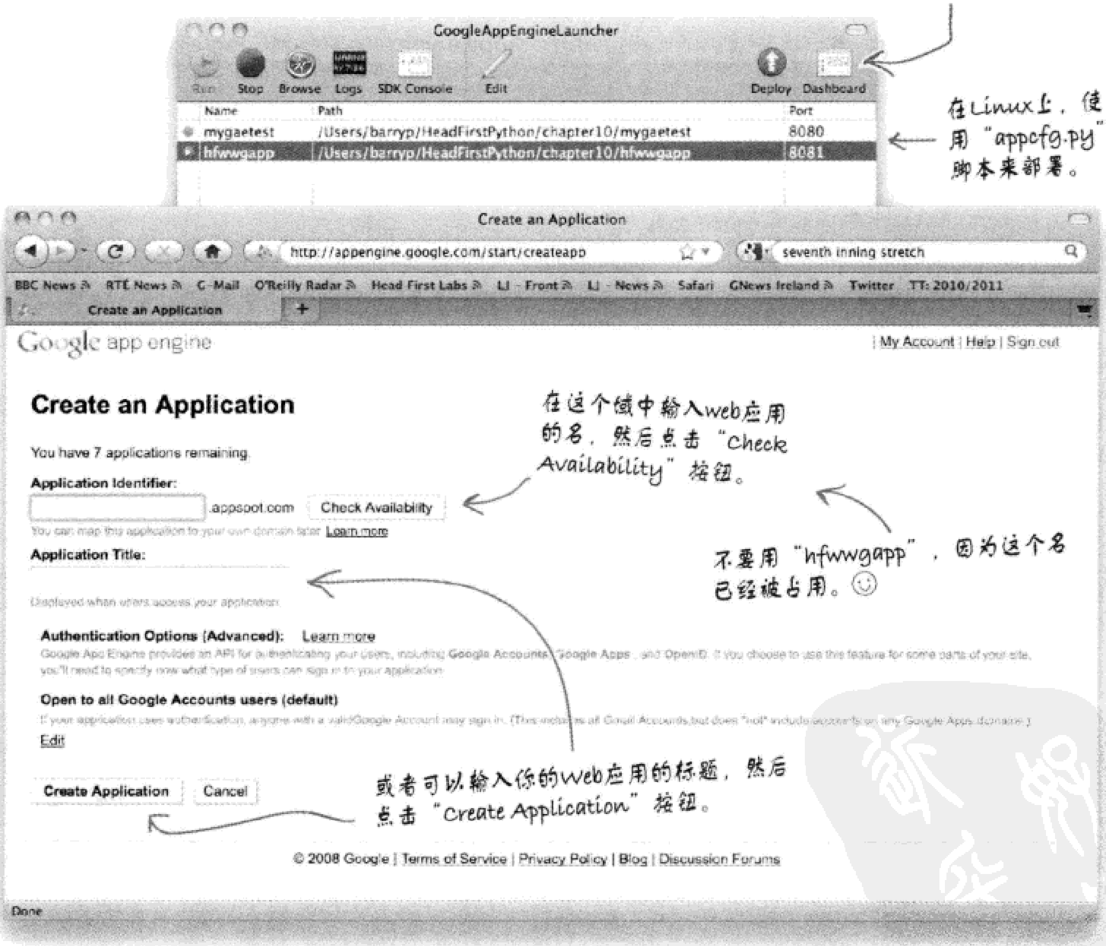
每次用户向datastore增加一个观鲸数据时，GAE会确保同时保存用户的Google Account ID。这个额外的标识信息使得HFWWG可以准确地跟踪是谁报告了哪一次观鲸记录，这样就（很有希望）大大减少Web应用收到的垃圾信息。

剩下的就是将这个Web应用部署到Google的云基础设施。

将Web应用部署到Google云

既然已经在本地开发和测试了你的Web应用，现在可以把它部署在Google云上。这个过程包括两个步骤：注册和上传。

要在Google云上注册你的Web应用，需要点击GAE Launcher上的Dashboard按钮。

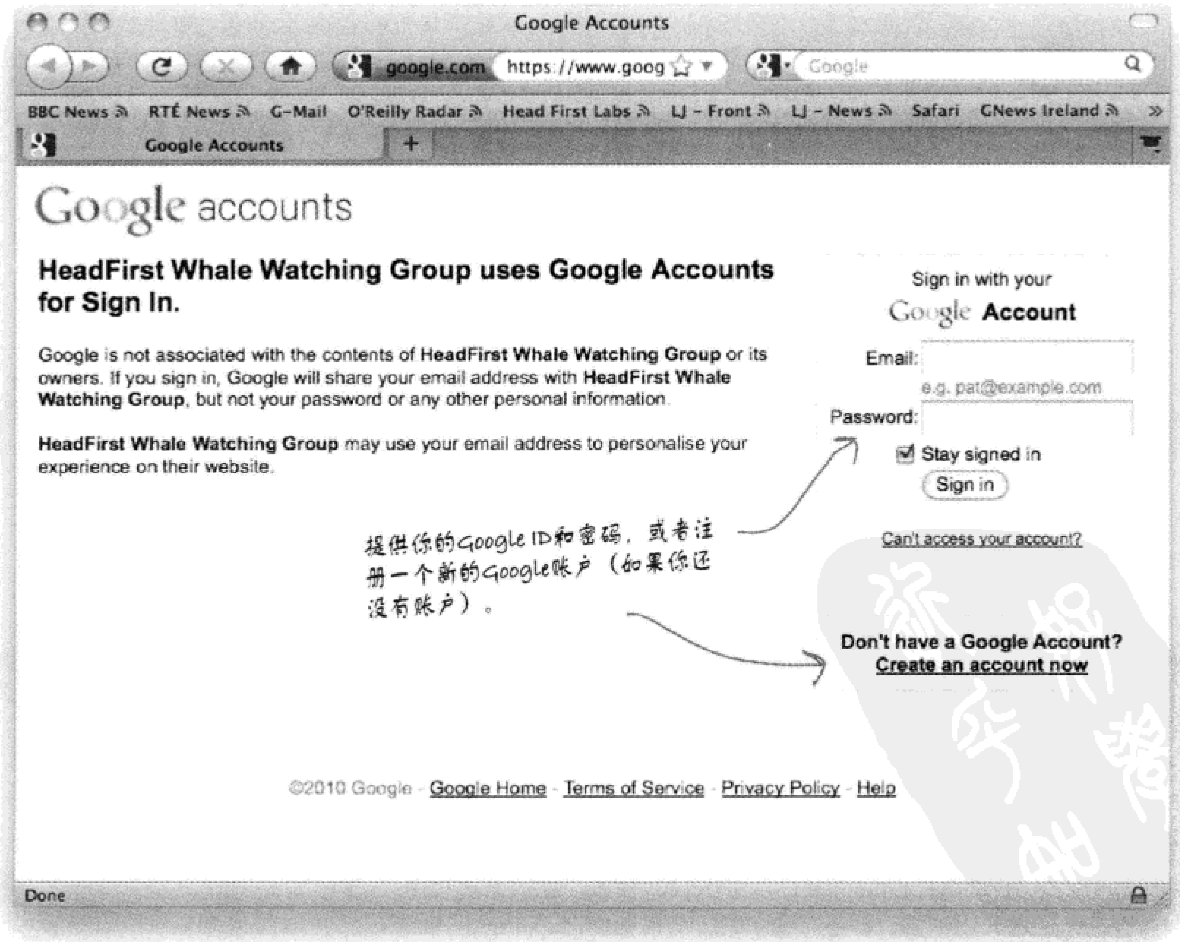


假设一切都按计划顺利进行，GAE确认你的应用已经创建，剩下的就是完成部署。返回到GAE Launcher，点击Deploy按钮。在部署进程中控制台会显示一堆状态消息。如果一切正常，会通知你"appcfg.py has finished with exit code 0"（appcfg.py成功完成，退出码为0）。

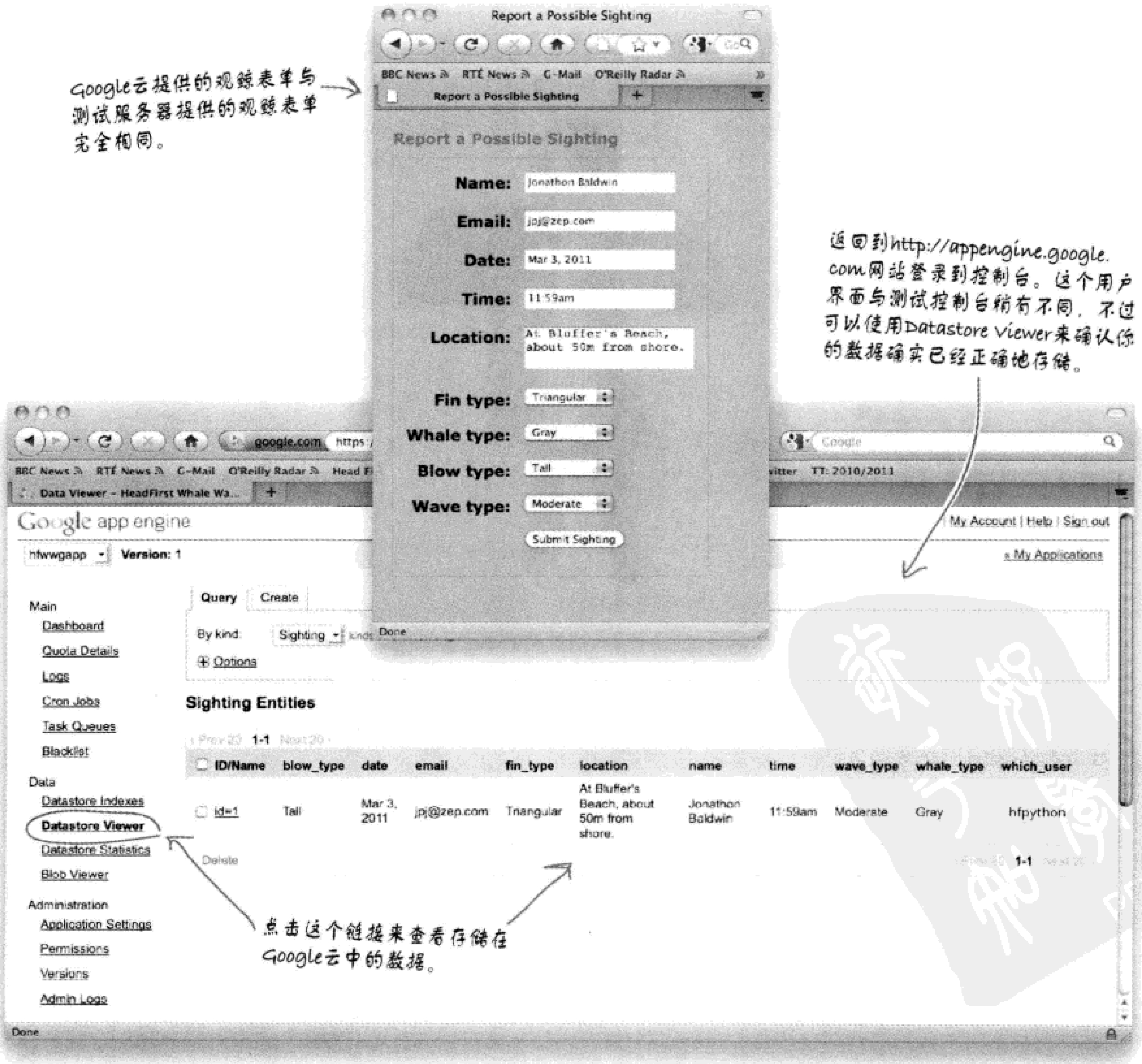
现在你的GAE Web应用可以在Google的云上运行了。

在Google上测试

下面让Web应用在Google的云上运行。打开你的Web浏览器，导航到一个以Web应用名开头并以*.appspot.com*结尾的Web地址。对于HFWWG Web应用，这个Web地址是*http://hfwwgapp.appspot.com*。第一次尝试访问Web应用时，App Engine会重定向到Google登录页面。



成功登录后，会出现观鲸表单。输入一些测试数据：



HFWWG Web应用已经成功部署！



简直太专业了。

你构建了一个很棒的数据输入Web应用，并把它部署在Google的云上。不管观鲸多频繁，无论每天观鲸的记录是曲指可数还是成百上千，你的Web应用都能很好地处理相应的负载，这要归功于Google的App Engine。而且，最好一点是，只要观鲸活动还没有达到每个月上百万次，财力紧张的HFWWG就根本不用付一分钱！

注意到了吗？你的所有代码都是用Python 2.5写的！



你的Python工具箱

你已经读完了第10章，并在你不断扩大的工具箱里增加了更多重要的Python工具。

App Engine术语

- "Datastore"——Google App Engine用来持久存储数据的数据存储库。
- "实体（Entity）"——对应"数据行"的一种称呼。
- "属性（Property）"——对应"数据值"的一种称呼。

- 每个App Engine Web应用必须有一个名为app.yaml的配置文件。
- 可以使用GAE Launcher启动、停止、监视、测试、上传和部署你的Web应用。
- App Engine的模板技术建立在Django项目所用模板技术基础之上。
- App Engine还可以使用Django的表单验证框架。
- 可以使用self.response对象来构造一个GAE Web响应。
- 可以使用self.request对象来访问一个GAE Web应用的表单数据。
- 响应一个GET请求时，要在一个get()方法中实现必要的功能。
- 响应一个POST请求时，要在一个post()方法实现必要的功能。
- 使用put()方法将数据存储在App Engine datastore中。

11 处理复杂性

数据加工



能在一个特定领域应用Python确实很棒。

不论是Web开发、数据库管理还是移动应用，Python都能帮助你完成任务，顺利地编写解决方案。不过除了这些问题，还存在另外一些问题：它们无法归类或关联到某一领域。这些问题本身很特别，必须用一种有区别的、非常特定的方式来考虑。为这些问题创建预定的软件解决方案正是Python擅长的领域。在最后这一章中，你的Python技能将会更上一层楼，从而能很好地解决这些问题。

下一次跑步有没有合适的目标时间？

Head First马拉松俱乐部（Head First Marathon Club）花了好几年时间收集和整理长跑者的数据。随着时间积累，俱乐部已经利用这些数据生成了一个很大的跑步数据电子表格，可以帮助长跑者预测他们跑不同距离时的表现。这个电子表格非常庞大，包含多达50列数据。

下面来看俱乐部的数据，并了解长跑者和他们的教练如何使用这些数据。

长跑者选择一个距离，比如说15公里，然后我们测定她跑这么长距离所用的时间。



这里给出马拉松俱乐部电子表格数据的一部分。

File Edit View Insert Format Form Tools Help

Formula: V02

	A	B	C	D	E	F
1	V02	84.8	82.9	81.1	79.3	77.5
2	2mi	8:00	8:10	8:21	8:33	8:44
3	5k	12:49	13:06	13:24	13:42	14:00
4	5mi	[illegible]	21:48	22:17	22:47	23:18
5	[illegible]	26:54	27:30	28:08	[illegible]	[illegible]
6	15k	41:31	42:27	43:24	44:23	45:23
7	[illegible]	44:46	45:46	46:48	47:51	[illegible]
8	20k	56:29	57:45	59:03	1:00:23	1:01:45
9	13.1mi	59:49	1:01:09	1:02:32	1:03:56	[illegible]
10	25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24
11	30k	1:27:10	1:19:08	1:31:08	1:33:11	[illegible]
12	Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16
13						

测定距离是15公里。

预测的跑马拉松的目标时间。

是的，她跑得确实很快！

我跑15公里用了45:01。我的教练在15k那一行查找与这个时间最接近的匹配值。



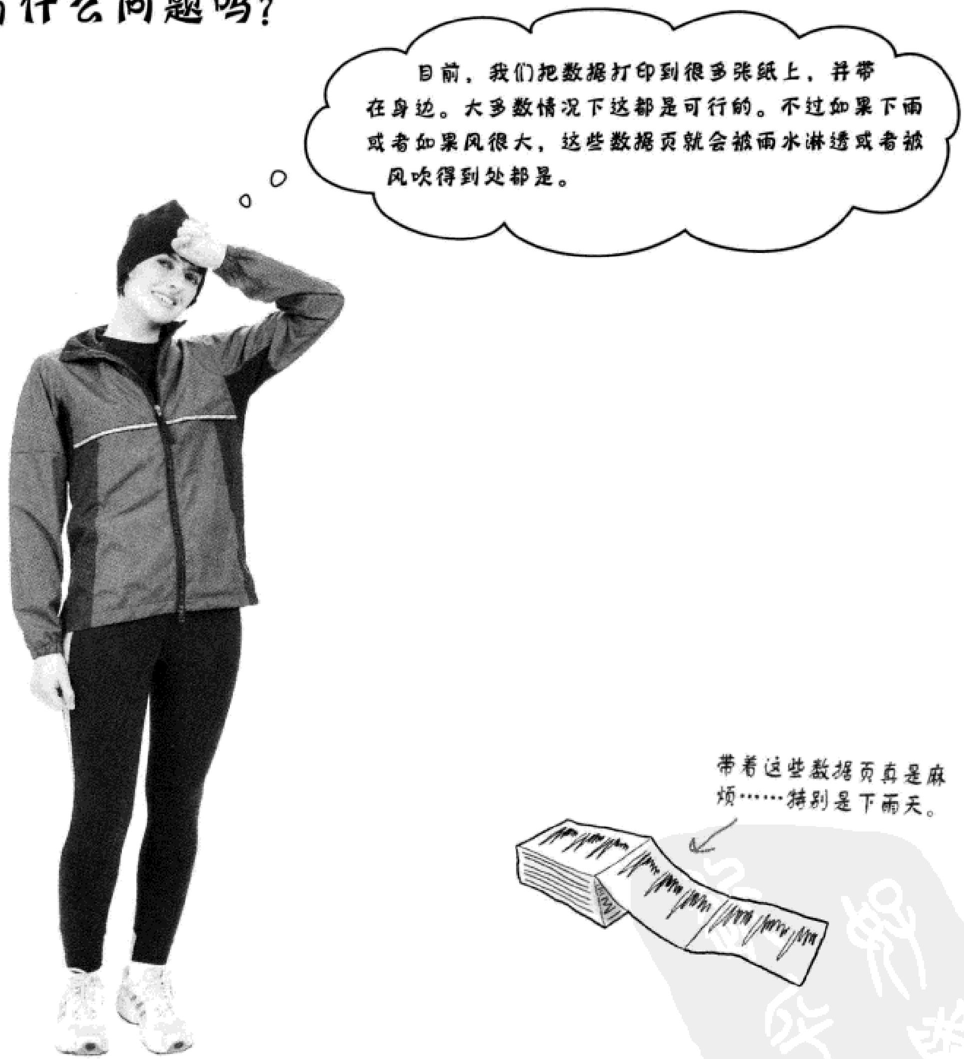
根据这个基准，我可以在列中上下查找，选择另外某个距离的目标时间，比如说跑马拉松的目标时间。



这个电子表格有些吓人……不过不用担心。很快你就会搞清楚的。

H	I	J	K	L	M	N	O
74.2	72.5	70.9	69.4	67.9	66.4	64.9	63.5
9:08	9:20	9:33	9:46	9:59	10:13	10:26	10:41
14:38	14:58	15:18	15:39	16:00	16:22	16:44	17:06
24:22	24:55	25:28	26:03	26:38	27:14	27:51	28:28
30:45	31:26	32:09	32:52	33:36	34:22	35:08	35:56
47:27	48:31	49:36	50:43	51:52	53:02	54:14	55:27
51:09	52:18	53:29	54:41	55:55	57:11	58:28	59:47
1:04:33	1:06:00	1:07:29	1:09:01	1:10:34	1:12:09	1:13:46	1:15:26
1:08:21	1:09:53	1:11:28	1:13:04	1:14:43	1:16:24	1:18:07	1:19:52
1:21:58	1:23:49	1:25:49	1:27:37	1:29:36	1:31:37	1:33:40	1:35:47
1:39:37	1:41:52	1:44:09	1:46:30	1:48:54	1:51:21	1:53:51	1:56:25
2:23:31	2:26:44	2:30:02	2:33:25	2:36:52	2:40:24	2:44:00	2:47:42

那么……有什么问题吗？



更何况还有另外一些麻烦，比如说可能忘了带这些数据页，还需要保证这些数据页是最新的，另外需要在这堆数据页中来回翻来找到最接近的匹配记录。

当然，你刚刚掌握的Python编程技能已经让你名声大噪，特别是在跑步者圈子里很有声誉。最理想地，马拉松俱乐部希望能有一个Android应用可以加载到各种手机上，这样只需把手机放在各个教练的口袋里（而不必整天扛着大堆的数据记录纸）。这个应用需要自动完成查找以及指定距离的时间预测。

你准备应对这个挑战吗？你认为自己能提供帮助吗？

从数据开始

对现在来说，先不要操心如何创建Android应用，很快就会解决这个问题。我们先来解决核心**数据加工问题**，等有了一个可行的解决方案后，再来考虑如何把这个解决方案移植到Android。首先将数据转换为一种能用Python轻松处理的格式。

大多数电子表格程序都能把数据导出为广泛使用的CSV格式。俱乐部已经为你完成了这个工作，创建了一个名为`PaceData.csv`的文件，其中包含原电子表格中各行的数据。

动手做！

从本书的支持网站下载`PaceData.csv`。

下面是这个CSV最前面的原始数据示例：

第一行数据是电子表格中的列标题。它们看起来像是数字，但实际上是标题，表示对应各列中跑步时间的估计最大耗氧量（或者称为VO2 Max，单位是ml/kg-min）。因为它们对计时数据没有任何影响，所以我们只是将其处理为标题。

```
V02,84.8,82.9,81.1,79.3,77.5,75.8,74.2,72.5,70.9,69.4,67.9,66.4,64.9,63.5,62.1,60.7,59.4,58.1,56.8,55.
2mi,8:00,8:10,8:21,8:33,8:44,8:56,9:08,9:20,9:33,9:46,9:59,10:13,10:26,10:41,10:55,11:10,11:25,11:40,1
5k,12:49,13:06,13:24,13:42,14:00,14:19,14:38,14:58,15:18,15:39,16:00,16:22,16:44,17:06,17:30,17:53,18:
```

其余各行的第一个值是测时的距离或行标签。

各行其余部分是记录的跑步时间的一个列表。

Sharpen your pencil

你必须以某种方式在Python程序中对CSV文件中的数据建立模型。你能想出对此有帮助的一种数据结构吗？并说明你选择这个数据结构的原因。

....

....

....

....

....

Sharpen your pencil Solution

你必须以某种方式在Python程序中对CSV文件中的数据建立模型。你能想出对此有帮助的一种数据结构吗？并说明你选择这个数据结构的原因。

标题列表可以存储在一个LIST中。

各行的时间列表也可以存储在一个LIST中，但是它们还需要与数据第一行中的标题关联，

所以这里实际需要的是不是DICTIONARY？

或者是不是需要这二者的某种结合？

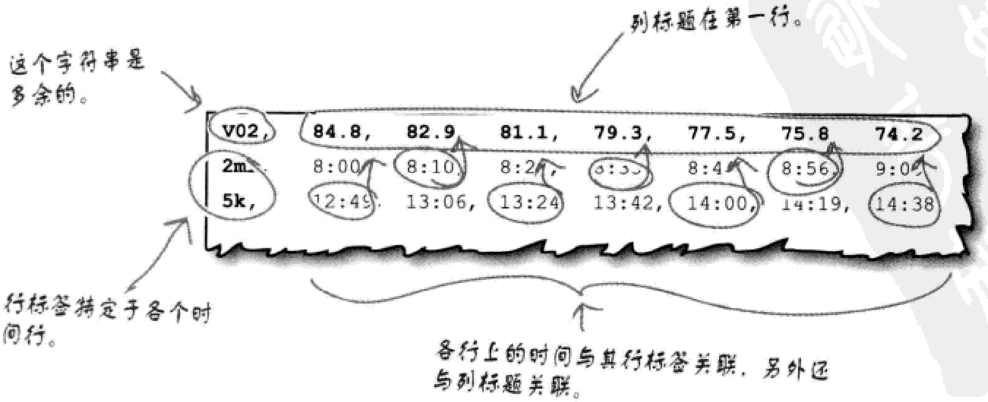
嗯……这里要考虑的问题很多。

再来查看数据

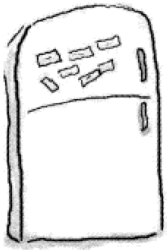
CSV文件中的第一行数据是**列标题**，这一行的第一个值（即字符串V02）是多余的，在应用的这个版本中从未用到。第一行其余的数据则是与各列时间值关联的标题。

当然，列中的数据还要与各行关联，这由第一列中的行**标签标识**，如2mi、5k等。

下面再来看CSV文件中的数据，这里已经重新格式化来帮助强调这些关联。



但是如何在代码中描述所有这些关联呢？



马拉松磁贴

下面是从CSV数据读取原始数据的一些代码。第一行中的列标题加载到一个名为column_headings的列表中。其余的数据（所有时间行）都加载到一个名为row_data的字典中，每个数据行以各行最前面的行**标签字符串**作为键。当然，不走运的事情总是经常发生，有人在清理冰箱，把一堆磁贴弄到地上了。看看你能不能按正确的顺序把这些磁贴重新摆好。

这里需要放什么？

```
with open('PaceData.csv') as paces:
```

这里要处理"column_headings"列表。

```
    for each_line in paces:
```

这里处理"row_data"。

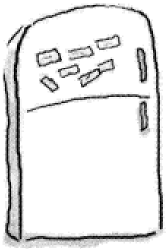
```
num_cols = len(column_headings)
print(num_cols, end=' -> ')
print(column_headings)

num_2mi = len(row_data['2mi'])
print(num_2mi, end=' -> ')
print(row_data['2mi'])

num_Marathon = len(row_data['Marathon'])
print(num_Marathon, end=' -> ')
print(row_data['Marathon'])
```

加载数据后，这些代码允许你检查是否一切正常。

```
row = each_line.strip().split(',')
column_headings =
row_label = row.pop(0)
row_data[row_label] = row
row_data = {}
paces.readline().strip().split(',')
column_headings.pop(0)
```



马拉松磁贴答案

下面是从CSV数据读取原始数据的一些代码。第一行中的列标题加载到一个名为column_headings的列表中。其余的数据（所有时间行）都加载到一个名为row_data的字典中，每个数据行以各行最前面的行**标签字符串**作为键。当然，不走运的事情总是经常发生，有人在清理冰箱，把一堆磁贴弄到地上了。看看你能不能按正确的顺序把这些磁贴重新摆好。

```
row_data = {}

with open('PaceData.csv') as paces:
    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)
    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        row_data[row_label] = row

num_cols = len(column_headings)
print(num_cols, end=' -> ')
print(column_headings)

num_2mi = len(row_data['2mi'])
print(num_2mi, end=' -> ')
print(row_data['2mi'])

num_Marathon = len(row_data['Marathon'])
print(num_Marathon, end=' -> ')
print(row_data['Marathon'])
```

要为各行时间创建一个空字典。

从第一行数据创建列标题。

从文件读取一行，从中去除不想要的空白符，然后按逗号分解。

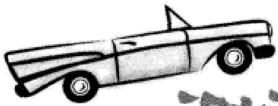
删除第一个标题，字符串“VO2”，你不需要这个数据。

处理文件的其余部分。

做同样的处理：读取数据行，去除空白符，再按逗号分解。

抽取行标签。

使用行标签并结合行中其余的数据来更新字典。



测试驱动

将代码加载到IDLE，CSV文件与代码在同一个文件夹中，运行这个代码看看屏幕上会显示什么。

将代码加载到IDLE。

marathon.py - /Users/barryp/HeadFirstPython/chapter11/marathon.py

```
row_data = {}

with open('PaceData.csv') as paces:

    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)

    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        row_data[row_label] = row

num_cols = len(column_headings)
print(num_cols, end=' -> ')
print(column_headings)

num_2mi = len(row_data['2mi'])
print(num_2mi, end=' -> ')
```

输出确认了每个数据行有50个数据项。

列标题。

"2mi"的相应数据行。

"Marat-hon"相应数据行。

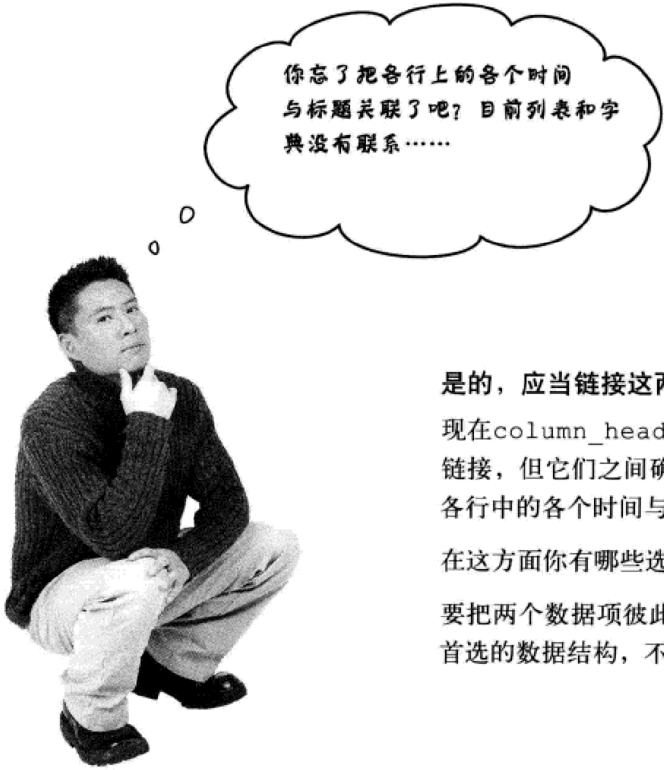
Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
50 -> ['84.8', '82.9', '81.1', '79.3', '77.5', '75.8', '74.2', '72.5', '70.9', '6
9.4', '67.9', '66.4', '64.9', '63.5', '62.1', '60.7', '59.4', '58.1', '56.8', '55
.5', '54.3', '53.1', '52', '50.8', '49.7', '48.6', '47.5', '46.5', '45.5', '44.5'
, '43.5', '42.5', '41.6', '40.7', '39.8', '38.9', '38', '37.2', '36.4', '35.6', '
34.8', '34', '33.3', '32.6', '31.8', '31.1', '30.5', '29.8', '29.1', '28.5']
50 -> ['8:00', '8:10', '8:21', '8:33', '8:44', '8:56', '9:08', '9:20', '9:33', '9
:46', '9:59', '10:13', '10:26', '10:41', '10:55', '11:10', '11:25', '11:40', '11:
56', '12:12', '12:29', '12:45', '13:03', '13:20', '13:38', '13:57', '14:16', '14:
35', '14:54', '15:15', '15:35', '15:56', '16:18', '16:40', '17:02', '17:25', '17:
49', '18:13', '18:38', '19:03', '19:28', '19:55', '20:22', '20:49', '21:17', '21:
46', '22:15', '22:45', '23:16', '23:48']
50 -> ['2:05:34', '2:08:24', '2:11:17', '2:14:15', '2:17:16', '2:20:21', '2:23:31
', '2:26:44', '2:30:02', '2:33:25', '2:36:52', '2:40:24', '2:44:00', '2:47:42', '
2:51:28', '2:55:20', '2:59:16', '3:03:18', '3:07:26', '3:11:39', '3:15:58', '3:20
:22', '3:24:53', '3:29:29', '3:34:12', '3:39:01', '3:43:57', '3:48:59', '3:54:09'
, '3:59:25', '4:04:48', '4:10:18', '4:15:56', '4:21:42', '4:27:35', '4:33:36', '4
:39:46', '4:46:04', '4:52:30', '4:59:05', '5:05:48', '5:12:41', '5:19:43', '5:26:
55', '5:34:16', '5:41:48', '5:49:29', '5:57:21', '6:05:23', '6:13:37']
>>>
```

Ln: 9 Col: 4

开局不错：你已经从CSV中读取了数据，把标题放入一个列表，并把数据放入一个字典。

接下来呢？



是的，应当链接这两个数据结构。

现在`column_headings`列表和`row_data`字典没有任何链接，但它们之间确实应该有关联。我们需要一种方法将各行中的各个时间与该数据列最上面的标题关联。

在这方面你有哪些选择？

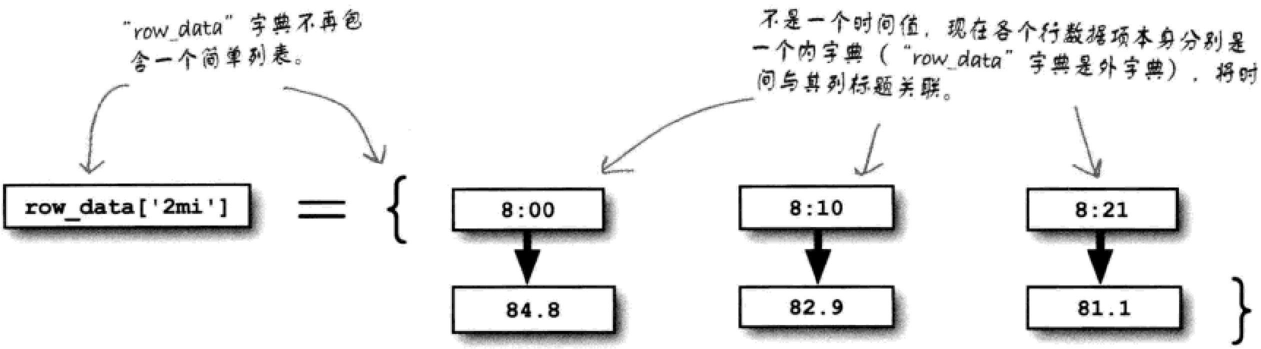
要把两个数据项彼此链接（或关联）起来，Python字典是首选的数据结构，不是吗？

PDG

将各个时间存储为字典

不要简单地把各个时间作为一个数存储在row_data字典中，下面把这些数据作为一个字典来存储，键设置为时间，值设置为列标题。这样一来，就可以快速而简便地确定与某个时间关联的列，对不对？

倘若存在这种关联，下面给出Python内存中数据结构的一个片段：



你要做的就是明确如何用行数据和关联的列标题填充这个内字典……你会得到所需的全部数据。

创建这个数据结构的技巧在于，要认识到各行（包括列标题）的大小是固定的：都包含50项。了解到这一点，创建所需要的字典就不难了。

```
row_data = {}
with open('PaceData.csv') as paces:
    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)
    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        inner_dict = {}
        for i in range(len(column_headings)):
            inner_dict[row[i]] = column_headings[i]
        row_data[row_label] = inner_dict
```

这里不需要做任何修改。

创建另一个空字典。

每次迭代时，"i"将是当前列号。

不再硬编码写入50，而是计算出这个值。

将列标题与行中的时间值关联。

填充字典后，把它赋至"row_data"中的相应行标签。

An IDLE Session

将以上字典填充代码增加到程序中。把程序最后的所有`print()`语句删除，因为你将使用IDLE shell来测试代码。按下`F5`或从Run菜单选择Run Module选项来运行代码。使用`dir()` BIF确认程序代码确实正常执行，并在Python的命名空间创建了一组变量：

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'column_headings', 'each_line', 'i',
'inner_dict', 'paces', 'row', 'row_data', 'row_label']
```

代码的所有变量都存在。

V02	84.8	82.9	81.1	79.3	77.5
2mi	8:00	8:10	8:21	8:33	8:44
5k	12:49	13:06	13:24	13:42	14:00
5mi	21:19	21:48	22:17	22:47	23:18
10k	26:54	27:30	28:08	28:45	29:24
15k	41:31	42:27	43:24	44:23	45:23
10mi	44:46	45:46	[illegible]	47:51	48:56
20k	56:29	57:45	59:03	1:00:23	1:01:45
13.1mi	59:49	1:01:09	1:02:32	1:03:56	1:05:23
25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24
30k	1:27:10	1:19:08	1:31:08	1:33:11	1:35:17
Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16

再来看以上的（部分）电子表格数据文件，下面尝试找出与15k行上的时间43:24相关联的列标题。然后使用这个列标题找到跑20公里的预测时间：

```
>>> column_heading = row_data['15k']['43:24']
>>> column_heading
'81.1'
```

关联的列标题正确地定义为“81.1”。

```
>>> prediction = [k for k in row_data['20k'].keys() if row_data['20k'][k] == column_heading]
>>> prediction
['59:03']
```

而且正确地预测出“59:03”。

预测代码剖析

下面花点时间来分析上一页IDLE会话最后发生了什么。这行代码是对存储在row_data中的“字典的字典”完成的一个双重字典查找：

这是”row_data”中的一个字典键。

这是另一个字典键，它是”row_data['15k']”字典中的索引。

```
column_heading = row_data['15k']['43:24']
```

查找'15k'行，然后查找与时间'43:24'关联的值，再将这个值赋至“column_heading”。

要得出20k相应数据行中的预测时间，为此需要在行字典中**查找某个键**，该键相应的值应设置为刚才找到的column_heading中存储的值。

你只对满足这个条件的数据感兴趣。

```
prediction = [k for k in row_data['20k'].keys() if row_data['20k'][k] == column_heading]
```

这就是你要找的数据。

这里非常适合使用一个条件列表推导。应该记得，列表推导语法就是for循环的一种简写记法。这个循环在与row_data['20k']字典关联的键列表中搜索数据。如果与键关联的值（k中的值）等于column_heading，k的值将增加到推导结果中，然后赋至一个名为predicton的新列表。

这个推导中确实完成了大量工作。



不要被列表推导吓住了。

应该记得，完全可以用一个等价的for循环重写列表推导……

嗯……有办法了。

PDG

Sharpen your pencil

使用for循环重写这一页上的各个列表推导。

```
times = [t for t in row_data['Marathon'].keys()]
```

```
headings = [h for h in sorted(row_data['10mi'].values(), reverse=True)]
```

```
time = [t for t in row_data['20k'].keys() if row_data['20k'][t] == '79.3']
```

Sharpen your pencil Solution

使用for循环重写这一页上的各个列表推导。

```
times = [t for t in row_data['Marathon'].keys()]
```

首先是一个空列表。

```
times = []
for each_t in row_data['Marathon'].keys():
    times.append(each_t)
```

将字典的键转换为一个列表。

每次迭代时，将键（一个时间值）追加到"times"列表。

```
headings = [h for h in sorted(row_data['10mi'].values(), reverse=True)]
```

首先是一个空列表。

```
headings = []
for each_h in sorted(row_data['10mi'].values(), reverse=True):
    headings.append(each_h)
```

将字典的值转换为一个列表……

……一定要按逆序（从大到小）对值排序。

每次迭代时，将值（一个列标题）追加到"times"列表。

```
time = [t for t in row_data['20k'].keys() if row_data['20k'][t] == '79.3']
```

首先是一个空列表。

```
time = []
for each_t in row_data['20k'].keys():
    if row_data['20k'][each_t] == '79.3':
        time.append(each_t)
```

将字典的键转换为一个列表。

这里出现一个确定的模式。:-)

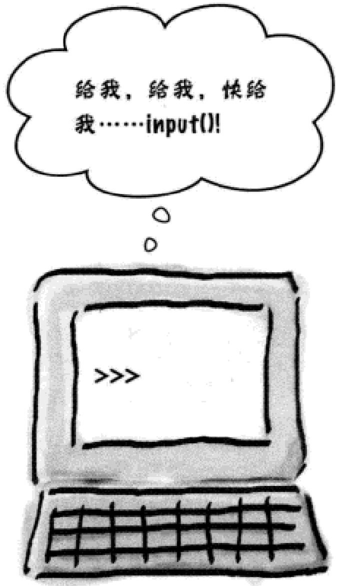
每次迭代时，查看列标题（字典的值部分）是否等于"79.3"，如果等于，则将时间追加到列表。

得到用户输入

既然数据已经放在一个Python数据结构中，现在该问问你的用户想要找什么。

具体来讲，你需要知道3个信息：跑步的距离、记录的时间，以及需要预测哪个距离的时间。

将你的应用移植到Android时，可以用一个漂亮的图形化对话框要求用户输入，不过，对于现在，先来快速创建一个基于文本的用户界面，利用它开发和测试应用所需的其余功能。完成这些工作之后，再来创建Android应用。



使用input()完成输入

Python有一个`input()` BIF可以提供帮助，这个BIF会在屏幕上显示一个提示语，然后接收键盘输入，并把输入的内容作为一个字符串返回给代码。

An IDLE Session

最好用例子来说明`input()` BIF的使用：

```
>>> res = input('What is your favorite programming language: ')
What is your favorite programming language: Python
>>> res
'Python'
```

提供提示语显示给用户。

输入的数据赋至"res"，这是一个字符串。

`input()` BIF返回一个字符串，而且已经去除了尾部的所有换行符（输入的字符串最后通常会包含换行符）。要注意重要的一点，不论你认为输入的是何种类型的数据，所有输入都作为一个字符串返回：

```
>>> age = input('What is your age: ')
What is your age: 21
>>> age
'21'
>>> int(age)
21
```

输入的数据赋至"age"，这是一个字符串（尽管你想将它作为一个数字来处理）。

[编辑的话：多好的年龄……太让人羡慕了！]

使用这个数据之前，先将输入转换为你需要的类型。

获取输入产生了一个问题……

使用input()来得到你需要的输入并不难。这里给出前面的代码，其中增加了3个input()调用来与用户交互。

marathon.py - /Users/barryp/HeadFirstPython/chapter11/marathon.py

```
row_data = {}

with open('PaceData.csv') as paces:

    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)

    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        inner_dict = {}
        for i in range(len(column_headings)):
            inner_dict[row[i]] = column_headings[i]
        row_data[row_label] = inner_dict

distance_run = input('Enter the distance attempted: ')
recorded_time = input('Enter the recorded time: ')
predicted_distance = input('Enter the distance you want a prediction for: ')
```

Ln: 20 Col: 0

这里不需要多作解释，因为用“input()”完成用户交互非常简单。

程序运行时，用户输入一些数据，看看会发生什么。

Python Shell

```
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 20k
Enter the recorded time: 59:59
Enter the distance you want a prediction for: Marathon
>>>
>>>
>>> row_data[distance_run][recorded_time]
Traceback (most recent call last):
  File "<pyshell#76>", line 1, in <module>
    row_data[distance_run][recorded_time]
KeyError: '59:59'
>>> |
```

Ln: 205 Col: 4

产生了一个“KeyError”异常……可是为什么会产生异常呢？

如果不在字典中，就无法找到。

row_data字典中的数据原先来自电子表格，并从CSV文件读入你的程序。

如果输入到recorded_time变量的数据值在这个字典中，那么一切正常，因为存在一个匹配。不过，如果输入到recorded_time变量的数据无法与字典中的任何值匹配，你就会得到一个KeyError。

那么在训练期间如何处理这个“问题”呢？



针对20公里输入的时间(59:59)落在跑步表格的这两个值之间。

10k	26:54	27:30	28:08	28:45	29:24
15k	41:31	42:27	43:24	44:23	45:23
10mi	44:46	45:46	[illegible]	47:51	48:56
20k	56:29	57:45	59:03	1:00:23	1:01:45
13.1mi	59:49	1:01:09	1:02:32	1:03:56	1:05:23
25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24
30k	1:27:10	1:19:08	1:31:08	1:33:11	1:35:17
Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16

搜索最接近的匹配

你要做的就是在数据行中搜索最接近的匹配，对不对？没想到吧，Head First代码审查小组认为他们有一些函数可以提供帮助。



这个代码在一个名为“find_it.py”的文件中，可以从本书支持网站下载这个文件。

find_it.py - /Users/barryp/HeadFirstPython/chapter11/find_it.py

```
def find_closest(look_for, target_data):

    def whats_the_difference(first, second):
        if first == second:
            return(0)
        elif first > second:
            return(first - second)
        else:
            return(second - first)

    max_diff = 9999999
    for each_thing in target_data:
        diff = whats_the_difference(each_thing, look_for)
        if diff == 0:
            found_it = each_thing
            break
        elif diff < max_diff:
            max_diff = diff
            found_it = each_thing
    return(found_it)
```

Ln: 22 Col: 0

“find_closest”函数完成一个简单的线性搜索，返回“target_data”中与“look_for”参数最接近的值。

这是一个嵌套函数的例子，这在Python中是允许的。给定两个值，这个函数会返回二者之差。

这可能不是最高效的搜索代码，不过确实可用。

PDG

An IDLE Session

下面测试`find_it.py`模块，确定它是否满足你的应用的需要。将这个模块加载到IDLE，然后按下F5或者从Run菜单选择Run Module：

```
>>> find_closest(3.3, [1.5, 2.5, 4.5, 5.2, 6])
2.5
>>> find_closest(3, [1, 5, 6])
1
>>> find_closest(3, [1, 3, 4, 6])
3
>>> find_closest(3.6, [1.5, 2.5, 4.5, 5.2, 6])
4.5
>>> find_closest(3, [1, 4, 6])
4
>>> find_closest(2.6, [1.5, 2.5, 4.5, 5.2, 6])
2.5
```

给定一个要查找的值，并提供一些目标数据，“find_closest”函数看起来确实能完成任务。

下面用与CSV数据更类似的一些数据来测试：

```
>>> find_closest('59:59', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    find_closest('59:59', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
  File "/Users/barryp/HeadFirstPython/chapter11/find_it.py", line 15, in find_closest
    if diff == 0:
  File "/Users/barryp/HeadFirstPython/chapter11/find_it.py", line 11, in whats_the_difference
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

唉呀！这里出现了严重的问题。

BRAIN POWER

你认为哪里出了问题？为什么在要求处理CSV文件中的数据时`find_closest()`函数崩溃了？

时间有问题

CSV文件中的数据是计时值的一个表示，而不是具体的数字，CSV中的值都是**字符串**。这对你来说很好，因为你了解这些表示的含义。但Python只是把这些数据看作是字符串。

将数据发送到`find_closest()`函数时，Python试图将字符串作为数字来处理，这就导致了混乱。一种可行的办法是将这些表示时间的字符串转换为数字。不过怎么做呢？



时间-秒转换模块

Head First代码审查小组实在太好心了。尽管这个tm2secs2tm.py模块的名字确实有些奇怪，但看来它很有帮助。

这是他们提供的"tm2secs2tm.py"模块。

从本书支持网站可以下载这个代码。

tm2secs2tm.py - /Users/barryp/HeadFirstPython/chapter11/tm2secs2tm.py

```
import time

def format_time(time_string):
    tlen = len(time_string)
    if tlen < 3:
        original_format = '%S'
    elif tlen < 6:
        original_format = '%M:%S'
    else:
        original_format = '%H:%M:%S'
    time_string = time.strftime('%H:%M:%S', time.strptime(time_string, original_format))
    return(time_string)

def time2secs(time_string):
    time_string = format_time(time_string)
    (hours, mins, secs) = time_string.split(':')
    seconds = int(secs) + (int(mins)*60) + (int(hours)*60*60)
    return(seconds)

def secs2time(seconds):
    return(time.strftime('%H:%M:%S', time.gmtime(seconds)))
```

Ln: 24 Col: 0

这个函数确保所有时间都采用"HH:MM:SS"格式。这样有助于保证将时间转换为秒数时更为简单。

给定一个"时间字符串"，把它转换为一个秒数值。

将一个秒数值转换为"时间字符串"。



既然有了tm2secs2tm.py和find_it.py模块，下面创建一个函数使用这些模块提供的功能来解决你的搜索问题。这个新函数名为find_nearest_time()，它取两个参数……要查找的时间以及所搜索的时间列表。这个函数将把找到的最接近的时间作为一个字符串返回：

已经为你写好了开始的代码。

```
from find_it import find_closest
from tm2secs2tm import time2secs, secs2time

def find_nearest_time(look_for, target_data):
```

与上一章不同，在这里只需4行代码完全可以做到你想做的事情。

....

....

....

....

Exercise Solution

既然有了tm2secs2tm.py和find_it.py模块，下面创建一个函数使用这些模块提供的功能来解决你的搜索问题。这个新函数名为find_nearest_time()，它取两个参数……要查找的时间以及所搜索的时间列表。这个函数将把找到的最接近的时间作为一个字符串返回：

导入代码审查小组的代码。

```
from find_it import find_closest
from tm2secs2tm import time2secs, secs2time

def find_nearest_time(look_for, target_data):
    what = time2secs(look_for)
    where = [time2secs(t) for t in target_data]
    res = find_closest(what, where)
    return(secs2time(res))
```

这个函数取两个参数，一个时间字符串，以及一个时间字符串列表。

将要查找的时间字符串转换为等价的秒数值。

将时间字符串行转换为秒数。

调用“find_closest()”，并提供已转换的数据。

转换回时间字符串后，将最接近的匹配结果返回给调用代码。

An IDLE Session

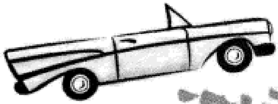
下面在IDLE shell尝试运行这个代码，看看时间“问题”是否已经解决：

以下是一些跑步数据。下面处理“20k”行的数据。

10k	26:54	27:30	28:08	28:45	29:24
15k	41:31	42:27	43:24	44:23	45:23
10mi	44:46	45:46	46:48	47:51	48:56
20k	56:29	57:45	59:03	1:00:23	1:01:45
13.1mi	59:49	1:01:09	1:02:32	1:03:56	1:05:23
25k	1:11:43	1:13:20	1:14:59	1:16:40	1:18:24
30k	1:27:10	1:19:08	1:31:08	1:33:11	1:35:17
Marathon	2:05:34	2:08:24	2:11:17	2:14:15	2:17:16

```
>>> find_nearest_time('59:59', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:00:23'
>>> find_nearest_time('1:01:01', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:00:23'
>>> find_nearest_time('1:02:01', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:01:45'
>>> find_nearest_time('57:06', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'00:56:29'
```

太棒了！看起来干得不错。



测试驱动

利用所有已有的代码，把它们集成在你的程序中，为马拉松俱乐部的预测问题生成一个完整的解决方案应该并不困难。下面就来测试运行这个程序。

marathon.py - /Users/barryp/HeadFirstPython/chapter11/marathon.py

```
from find_it import find_closest
from tm2secs2tm import time2secs, secs2time

def find_nearest_time(look_for, target_data):
    what = time2secs(look_for)
    where = [time2secs(t) for t in target_data]
    res = find_closest(what, where)
    return(secs2time(res))

row_data = {}

with open('PaceData.csv') as paces:

    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)

    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        inner_dict = {}
        for i in range(len(column_headings)):
            inner_dict[row[i]] = column_headings[i]
        row_data[row_label] = inner_dict

distance_run = input('Enter the distance attempted: ')
recorded_time = input('Enter the recorded time: ')
predicted_distance = input('Enter the distance you want a prediction for: ')

closest_time = find_nearest_time(recorded_time, row_data[distance_run])
closest_column_heading = row_data[distance_run][closest_time]

prediction = [k for k in row_data[predicted_distance].keys()
                    if row_data[predicted_distance][k] == closest_column_heading]

print('The predicited time running ' + predicted_distance + ' is: ' + prediction[0] + '.')
```

Ln: 37 Col: 0

这个代码"原样"使用，未做改动。

在数据中查找最接近的时间。

抽取列标题。

搜索指定距离的一个预测时间，并在屏幕上显示。

Python Shell

```
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 20k
Enter the recorded time: 59:59
Enter the distance you want a prediction for: Marathon
Traceback (most recent call last):
  File "/Users/barryp/HeadFirstPython/chapter11/marathon.py", line 31, in <module>
    closest_column_heading = row_data[distance_run][closest_time]
KeyError: '01:00:23'
>>>
```

Ln: 13 Col: 4

用前面同样的数据输入来测试这个程序。

又产生一个"KeyError"……

做了所有这些工作后，你又得到了同样的错误。真是郁闷。

时间还有问题……

或者更确切地讲，tm2secs2tm.py模块对时间字符串格式化的方式还有问题。再来看上一个IDLE会话的结果。注意到find_nearest_time()函数调用返回的结果有什么奇怪的地方吗？

```
>>> find_nearest_time('59:59', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:00:23'
>>> find_nearest_time('1:01:01', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:00:23'
>>> find_nearest_time('1:02:01', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'01:01:45'
>>> find_nearest_time('57:06', ['56:29', '57:45', '59:03', '1:00:23', '1:01:45'])
'00:56:29'
```

返回的所有时间都采用“HH:MM:SS”格式。

代码得到返回的某个值时，尝试在字典中索引，但是没有找到匹配，因为字典的键不符合HH:MM:SS格式。要解决这个问题，解决方案就是确保每次在代码中使用一个时间字符串时都要确保它采用HH:MM:SS格式：

marathon.py - /Users/barryp/HeadFirstPython/chapter11/marathon.py

```
from find_it import find_closest
from tm2secs2tm import time2secs, secs2time, format_time

def find_nearest_time(look_for, target_data):
    what = time2secs(look_for)
    where = [time2secs(t) for t in target_data]
    res = find_closest(what, where)
    return(secs2time(res))

row_data = {}

with open('PaceData.csv') as paces:

    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)

    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        inner_dict = {}
        for i in range(len(column_headings)):
            inner_dict[format_time(row[i])] = column_headings[i]
        row_data[row_label] = inner_dict

distance_run = input('Enter the distance attempted: ')
recorded_time = input('Enter the recorded time: ')
predicted_distance = input('Enter the distance you want a prediction for: ')

closest_time = find_nearest_time(format_time(recorded_time), row_data[distance_run])
closest_column_heading = row_data[distance_run][closest_time]

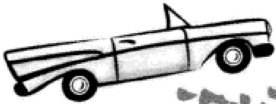
prediction = [k for k in row_data[predicted_distance].keys()
                if row_data[predicted_distance][k] == closest_column_heading]

print('The predicited time running ' + predicted_distance + ' is: ' + prediction[0] + '.')
```

Ln: 37 Col: 0

从“tm2secs2tm.py”模块导入“format_time()”函数。

使用这个函数确保代码内部使用的时间采用“HH:MM:SS”格式。



测试驱动

下面再尝试运行一次代码。既然现在系统中所有时间字符串都符合HH:MM:SS格式，希望这一次代码能成功运行。

Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 20k
Enter the recorded time: 59:59
Enter the distance you want a prediction for: Marathon
Traceback (most recent call last):
  File "/Users/barryp/HeadFirstPython/chapter11/marathon.py", line 31, in <module>
    closest_column_heading = row_data[distance_run][closest_time]
KeyError: '01:00:23'
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 20k
Enter the recorded time: 59:59
Enter the distance you want a prediction for: Marathon
The predicited time running Marathon is: 02:14:15.
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 5mi
Enter the recorded time: 23:45
Enter the distance you want a prediction for: 10mi
The predicited time running 10mi is: 00:50:02.
>>> ================================ RESTART ================================
>>>
Enter the distance attempted: 10k
Enter the recorded time: 32:15
Enter the distance you want a prediction for: 25k
The predicited time running 25k is: 01:25:49.
>>>
```

Ln: 31 Col: 4

这是前一次测试，报“KeyError”错误并崩溃。

这一次程序表现正常，可以成功运行。

另一次测试，确认程序确实能正常工作。

最后一次测试，确保程序无误。

现在能正常工作了。你已经解决了应用的核心问题：程序从CSV文件读入电子表格数据，将数据转换为一个“字典的字典”，允许与用户交互来得到对应某个特定距离记录的跑步时间，然后预测跑另一个距离可能需要的时间。

如果不算Head First代码审查小组提供的代码，你只需不到40行代码就解决了这个问题。这确实是一个不小的成就。剩下的就是将你的程序移植到俱乐部的Android手机上。

移植到Android并不需要太长时间，对吧？

移植到Android

你的代码已经能很好地工作。现在要把这个基于文本的Python程序移植到Android。大多数代码都不需要做任何改动，只需要修改与用户交互的部分。

显然，你希望这个最新Android应用的用户使用应用时要尽可能简单，可以提供类似下面的界面。



你的Android应用就是一堆对话框

这个Android应用通过一系列对话框与用户交互。除了请求用户提供数据的对话框，另外三个对话框有一些相似性。可以创建一个工具函数抽象出对话框创建的细节，从而利用这些共同的特性：

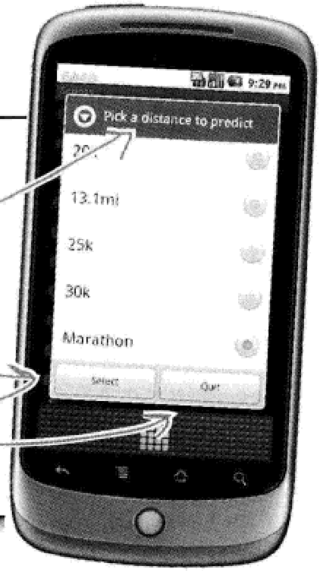
对话框的标题字符串。

要显示的值列表。

对话框创建方法名。

按钮的文本，有缺省值。

```
def do_dialog(title, data, func, ok='Select', notok='Quit'):
    app.dialogCreateAlert(title)
    func(data)
    app.dialogSetPositiveButtonText(ok)
    if notok:
        app.dialogSetNegativeButtonText(notok)
    app.dialogShow()
    return(app.dialogGetResponse().result)
```



显示对话框，然后返回所选择的项。

Sharpen your pencil

假设存在一个名为distances的列表，其中包含行距离标签（2mi, 5k, 5mi等）。在下面给出的空白处，提供两个必要的do_dialog()函数调用来创建上一页左边显示的两个dialogSetSingleChoiceItems。

Sharpen your pencil Solution

假设存在一个名为distances的列表，其中包含行距离标签（2mi, 5k, 5mi等）。在下面给出的空白处，提供两个必要的do_dialog()函数调用来创建上一页左边显示的两个dialogSetSingleChoiceItems。

```
do_dialog("Pick a distance", distances,
                app.dialogSetSingleChoiceItems)
do_dialog("Pick a distance to predict", distances,
                app.dialogSetSingleChoiceItems)
do_dialog('The predicited time running ' + predicted_distance +
                ' is: ', prediction, app.dialogSetItems, "OK", None)
```

提供要显示的项列表。

提供对话框标题。

提供要使用的对话框类型。

同上：对其他对话框做同样的工作。

这是另一个例子。

最后一个稍有点技巧，因为你要根据一些变量来建立对话框标题（所以首先需要创建这些变量）。

这一次使用一个不同的对话框创建方法。

覆盖对话框按钮的缺省值。

准备Android应用的代码

要使用以上对话框创建代码，需要导入必要的库，定义一些常量，创建一个Android对象，并重用本书前面的一些代码：

```
import time
import android
    ...
distances = [ '2mi', '5k', '5mi', '10k', '15k', '10mi', '20k',
              '13.1mi', '25k', '30k', 'Marathon' ]
    ...
hello_msg = "Welcome to the Marathon Club's App"
quit_msg = "Quitting the Marathon Club's App."
    ...
app = android.Android()

def status_update(msg, how_long=2):
    app.makeToast(msg)
    time.sleep(how_long)
```

完成导入。

创建一个行标签列表。

定义两个友好的消息。

创建一个Android应用对象。

这是本书前面的一个函数，在这里"原样"使用。

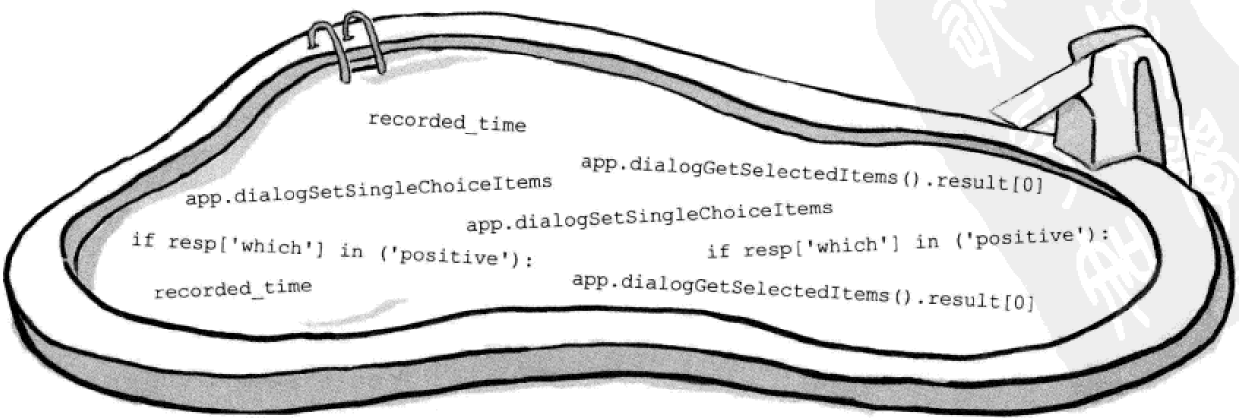
Android池塘谜题



你的任务是从这个池塘中取出代码，放在Android应用代码中的空行上。可以假设row_data字典已经存在，并且已经填入数据。上一页最后显示的变量也已经创建，另外可以使用status_update() 和do_dialog()函数。你的目标是摆放这些代码，来实现你需要的UI交互。

```
status_update(hello_msg)
resp = do_dialog("Pick a distance", distances, ............................... )
.................................................................
    distance_run = ......................................
    distance_run = distances[distance_run]
    ............ = app.dialogGetInput("Enter a " + distance_run + " time:",
                                      "Use HH:MM:SS format:").result
    closest_time = find_nearest_time(format_time( ............ ), row_data[distance_run])
    closest_column_heading = row_data[distance_run][closest_time]
    resp = do_dialog("Pick a distance to predict", distances, ............................... )
    .................................................................
        predicted_distance = ......................................
        predicted_distance = distances[predicted_distance]
        prediction = [k for k in row_data[predicted_distance].keys()
                      if row_data[predicted_distance][k] == closest_column_heading]
        do_dialog('The predicted time running ' + predicted_distance + ' is: ',
                  prediction, app.dialogSetItems, "OK", None)
status_update(quit_msg)
```

dialogGetInput() 方法显示输入对话框。



Android池塘谜题答案



你的任务是从这个池塘中取出代码，放在Android应用代码中的空行上。可以假设row_data字典已经存在，并且已经填入数据。上一页最后显示的变量也已经创建，另外可以使用status_update() 和do_dialog()函数。你的目标是摆放这些代码，来实现你需要的UI交互。

```
status_update(hello_msg)
resp = do_dialog("Pick a distance", distances, app.dialogSetSingleChoiceItems)
if resp['which'] in ('positive'):
    distance_run = app.dialogGetSelectedItems().result[0]
    distance_run = distances[distance_run]
    recorded_time = app.dialogGetInput("Enter a " + distance_run + " time:",
                                       "Use HH:MM:SS format:").result
    closest_time = find_nearest_time(format_time(recorded_time), row_data[distance_run])
    closest_column_heading = row_data[distance_run][closest_time]
    resp = do_dialog("Pick a distance to predict", distances, app.dialogSetSingleChoiceItems)
    if resp['which'] in ('positive'):
        predicted_distance = app.dialogGetSelectedItems().result[0]
        predicted_distance = distances[predicted_distance]
        prediction = [k for k in row_data[predicted_distance].keys()
                          if row_data[predicted_distance][k] == closest_column_heading]
        do_dialog('The predicted time running ' + predicted_distance + ' is: ',
                          prediction, app.dialogSetItems, "OK", None)
status_update(quit_msg)
```

要求用户从标签列表中选择一个距离。

将选择的距离标签赋至"distance_run"。

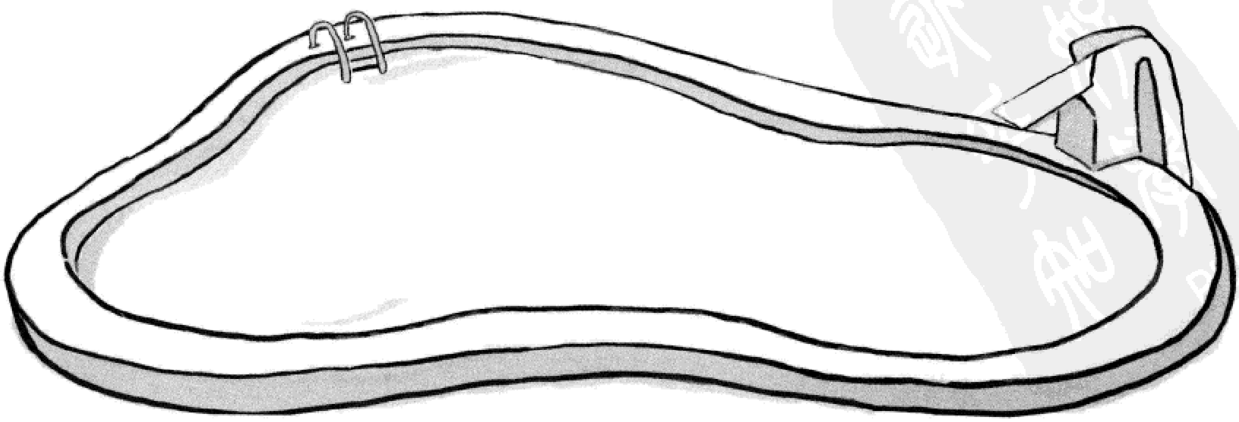
要求用户输入记录的时间。

得出要使用哪个列标题。

要求用户从标签列表中选择一个要预测的距离。

查找预测时间。

为用户显示所选距离的预测时间。



集成应用……

现在已经有了创建这个应用所需的全部代码：

marathonapp.py - /Users/barryp/HeadFirstPython/chapter11/marathonapp.py

```
import time
import android

from find_it import find_closest
from tm2secs2tm import time2secs, secs2time, format_time

def find_nearest_time(look_for, target_data):
    what = time2secs(look_for)
    where = [time2secs(t) for t in target_data]
    res = find_closest(what, where)
    return(secs2time(res))

distances = [ '2mi', '5k', '5mi', '10k', '15k', '10mi', '20k',
              '13.1mi', '25k', '30k', 'Marathon' ]
hello_msg = "Welcome to the Marathon Club's App"
quit_msg = "Quitting the Marathon Club's App."

row_data = {}

with open('/sdcard/sl4a/scripts/PaceData.csv') as paces:
    column_headings = paces.readline().strip().split(',')
    column_headings.pop(0)
    for each_line in paces:
        row = each_line.strip().split(',')
        row_label = row.pop(0)
        inner_dict = {}
        for i in range(len(column_headings)):
            inner_dict[format_time(row[i])] = column_headings[i]
        row_data[row_label] = inner_dict

app = android.Android()

def status_update(msg, how_long=2):
    app.makeToast(msg)
    time.sleep(how_long)

def do_dialog(title, data, func, ok='Select', notok='Quit'):
    app.dialogCreateAlert(title)
    func(data)
    app.dialogSetPositiveButtonText(ok)
    if notok:
        app.dialogSetNegativeButtonText(notok)
    app.dialogShow()
    return(app.dialogGetResponse().result)

status_update(hello_msg)

resp = do_dialog("Pick a distance", distances, app.dialogSetSingleChoiceItems)
if resp['which'] in ('positive'):
    distance_run = app.dialogGetSelectedItems().result[0]
    distance_run = distances[distance_run]
    recorded_time = app.dialogGetInput("Enter a " + distance_run + " time:",
                                       "Use HH:MM:SS format:").result
    closest_time = find_nearest_time(format_time(recorded_time), row_data[distance_run])
    closest_column_heading = row_data[distance_run][closest_time]
    resp = do_dialog("Pick a distance to predict", distances, app.dialogSetSingleChoiceItems)
    if resp['which'] in ('positive'):
        predicted_distance = app.dialogGetSelectedItems().result[0]
        predicted_distance = distances[predicted_distance]
        prediction = [k for k in row_data[predicted_distance].keys()
                        if row_data[predicted_distance][k] == closest_column_heading]
        do_dialog('The predicted time running ' + predicted_distance + ' is: ',
                  prediction, app.dialogSetItems, "OK", None)

status_update(quit_msg)
```

Ln: 67 Col: 0

完成导入。

包含“find_nearest()”函数。

声明常量。

注意：数据文件在SDCARD上的位置要特定于Android。

得到并预处理CSV数据。

创建Android应用对象，并包含你的辅助函数。

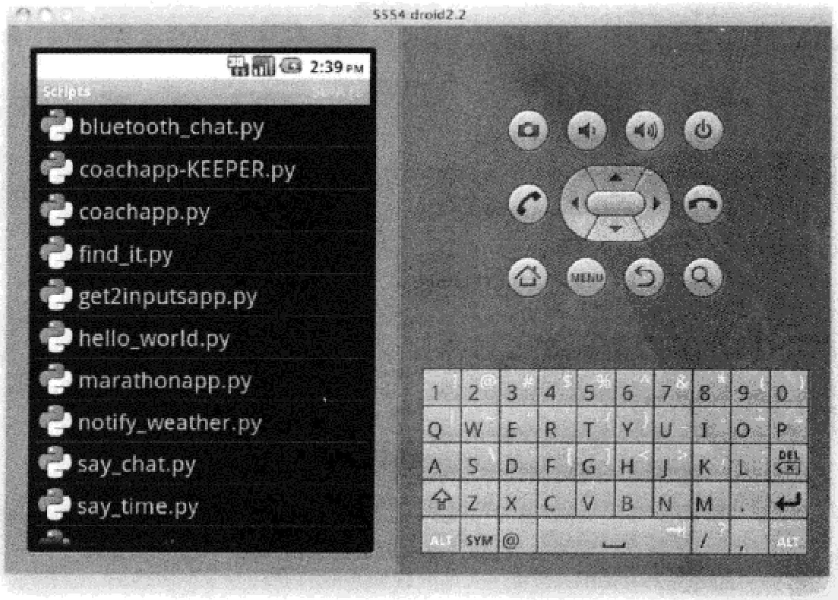
向用户显示用户界面，并处理交互。

测试驱动

把一个可以正常运行的应用加载到"真正"的手机上之前，首先在Android模拟器上测试这个Android应用。启动Android模拟器，先将你的代码以及代码需要的文件传输到模拟器的SDCARD。使用`tools`文件夹中的adb命令将`marathonapp.py`、`find_it.py`、`tm2sec2tm.py`和`PaceData.csv`复制到模拟器，然后尝试运行你的应用。

```
File Edit Window Help CopyToEmulator
$ tools/adb push marathonapp.py /mnt/sdcard/sl4a/scripts
43 KB/s (2525 bytes in 0.056s)
$ tools/adb push find_it.py /mnt/sdcard/sl4a/scripts
7 KB/s (555 bytes in 0.069s)
$ tools/adb push tm2secs2tm.py /mnt/sdcard/sl4a/scripts
12 KB/s (628 bytes in 0.050s)
$ tools/adb push PaceData.csv /mnt/sdcard/sl4a/scripts
59 KB/s (4250 bytes in 0.069s)
```

用这些命令将代码及其支持文件复制到模拟器。



请看……正等你来测试。

继续。你知道要做什么：点击这个应用！

应用大功告成!

剩下的就是把这个可以正常工作的Android应用移植到马拉松俱乐部的手机上……如果使用AndFTP这会很容易。当你展示你的最新成果时，俱乐部的成员们肯定无法相信他们的眼睛。



而且也没有什么能够阻碍你！

你的Python技艺已经在这里得到很好的应用和展示。

不论是为最小的手持设备，还是最大的Web服务器构建应用，你掌握的Python技术都能帮助你顺利完成任务。

祝贺你！



你的Python工具箱

你已经读完了第11章，现在已经拥有了一个完备的Python工具箱。祝贺你，好样的！

Python术语

- "条件"列表推导尾部包括一个"if"语句，允许在推导运行时控制哪些项可以增加到新列表中。
- 列表推导可以重写为一个等价的"for"循环。

BULLET POINTS

- `input()` BIF允许为用户提供提示语并接收输入。
- 如果你在使用Python 2而且需要`input()`函数，那么可以使用`raw_input()`函数。
- 可以结合Python内置的列表、集合和字典构造复杂的数据结构。
- `time`模块作为标准库的一部分，包含了大量函数可以完成时间格式之间的转换。

该分手了……



这还只是开始

很遗憾要说再见了，不过最重要的是将你所学付诸实用。你的Python之旅刚刚开始，现在要由你来掌握前行的方向。我们非常期待听到你的进展，所以经常来Head First Labs网站（***www.headfirstlabs.com***）看看，告诉我们Python对你的帮助！

附录：其他

(我们没有谈到的)十大问题



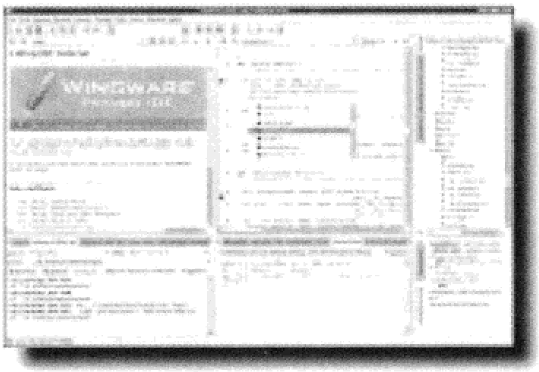
你已经学到了不少。

不过学习Python是永无止境的。你编写的Python代码越多，就越需要了解新的方法来处理某些事情。你还要掌握新工具和新技术。这本书实在篇幅有限，无法向你全面展示关于Python所需了解的一切。所以，下面给出我们没有谈到的十大问题，你可能希望下一步对这些问题有更多了解。

#1: 使用一个"专业"IDE

这本书中我们一直在使用Python的IDLE，刚开始学习Python时这个工具确实很合适，尽管有些古怪，但它可以处理大多数编程任务。它甚至还提供了一个内置的调试工具（查看Debug菜单），而且让人很难以置信，这个调试工具还相当先进。不过，你迟早都会需要一个功能更完备的集成开发环境。

WingWare Python IDE就是这样一个值得考虑的工具。这个专业级开发工具专门面向Python程序员，它由Python程序编写，也由Python程序员维护，而且它本身正是用Python编写的。WingWare Python IDE提供了多种许可方式：如果你是一个学生，或者在参与一个开源项目，这个工具就是免费的；不过如果你在一个以盈利为目的开发环境工作，则需要支付费用。



由Python程序员用Python为其他Python程序员编写……有了它，夫复何求？

WingWare Python IDE

另外还有一些更通用的工具。如果你在运行Linux，可以使用KDevelop IDE，它能与Python很好地集成。

当然，还有很多编程编辑器，通常你可能就需要这样一个编辑器。很多Mac OS X程序员就非常信赖TextMate编辑器。还有很多Python程序员在使用emacs和vi（或者更常见的变种vim）。本书作者就对vim非常着迷，另外每天也会大量使用IDLE和Python shell。

#2: 处理作用域

考虑下面这个简短的程序：

一个名为"name"的全局变量。

scope.py - /Users/barryp/HeadFirstPython/Top Ten Things/scope.py

```
name = "Head First Python"

def what_happens_here():

    print(name)
    name = name + " is a great book!"
    print(name)

what_happens_here()
print(name)
```

Ln: 12 Col: 0

一个读写全局变量"name"的函数。

调用这个函数。

看看函数运行之后"name"会设置成什么。

如果试图运行这个程序，Python会报一个错误消息：UnboundLocalError: local variable 'name' referenced before assignment（UnboundLocalError：局部变量'name'未赋值即被引用）……什么意思?

谈到作用域，Python允许你在函数中访问和读取一个全局变量的值，**但是不能修改**。Python看到这里的赋值时，它会查找一个名为name的局部变量，但是找不到，所以会抛出一个UnboundLocalError异常。要访问和修改一个全局变量，必须明确地表明你的意愿，如下所示：

scope.py - /Users/barryp/HeadFirstPython/Top Ten Things/scope.py

```
name = "Head First Python"

def what_happens_here():

    print(name)
    global name
    name = name + " is a great book!"
    print(name)

what_happens_here()
print(name)
```

Ln: 13 Col: 0

有些程序员对这种做法很不以为然，认为这很"丑陋"。还有些程序员认为这肯定是Monty Python设计这个编程语言时碰巧想到的。不管大家怎么想，我们现在只能这么做！

#3：测试

写代码是一方面，另一方面是**测试**。Python shell和IDLE的组合非常适合对小的代码片段进行测试和试验，不过对于规模更大的程序，测试框架则必不可少。

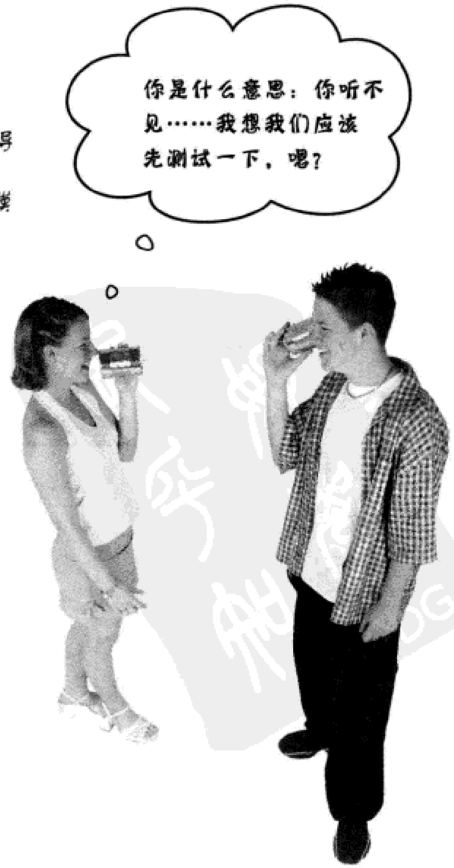
Python提供了两个测试框架。

对于从另一种现代编程语言转向Python的程序员来说，可能对第一个框架很熟悉，因为它基于流行的xUnit测试框架。Python的`unittest`模块（标准库的一部分）允许你为模块创建测试代码、测试数据和一个测试套件，而且与你的代码分开存放，即放在单独的文件中，从而允许采用多种不同方式测试你的代码。如果你在使用与当前语言类似的框架，那么可以放心，Python的实现基本上是相同的。

另一个测试框架名为`doctest`，这也是标准库的一部分。这个框架允许你从一个Python shell或IDLE会话取得输出，并用于测试。你要做的就是从shell复制内容，把它增加到你的模块文档串中。如果在模块最后增加类似下面的代码，它们将得到"doctest测试"：

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

如果代码作为一个模块导入，这段代码不会运行。如果从命令行运行你的模块，则会运行测试。



如果在操作系统命令行运行你的模块，测试会运行。如果你只想导入模块的代码，并不运行测试，那么前面的`if`语句会支持你做到这一点。

关于`unittest`和`doctest`的更多内容，请在网上搜索联机Python文档，或者通过IDLE的Help菜单来了解。

#4: 高级语言特性

对于这样一本书，我们知道除非页数是现在的3倍，否则绝对不可能全面介绍Python语言的所有内容。

而且要知道，即使我们确实涵盖了Python语言的方方面面，也没有人会感谢我们！

Python的内容还有很多，随着你的信心的增长，可以花些时间来研究这些高级语言特性：

匿名函数：lambda表达式（λ表达式）允许你创建简短的单行匿名函数（无函数名），一旦了解了匿名函数如何工作，它们会极其有用。

生成器：类似于迭代器，生成器允许处理数据序列。不同于迭代器的是，生成器通过使用yield表达式，可以尽量减少程序耗费的内存，同时还能在大规模数据集上提供类似迭代器的功能。

定义异常：基于Python作为标准提供的异常，可以创建你自己的异常对象。

函数修饰符：通过介入函数的启动和关闭机制，调整一个已存在函数的行为。

元类：这些定制类本身可以创建定制类。必须有一定胆量才敢使用元类，不过第10章中使用Django表单验证框架创建观鲸表单时确实使用了一个元类。

以上大多数（但不是全部）语言特性主要是一些高级Python程序员才感兴趣，他们需要构建由其他Python程序员使用的工具和语言扩展。

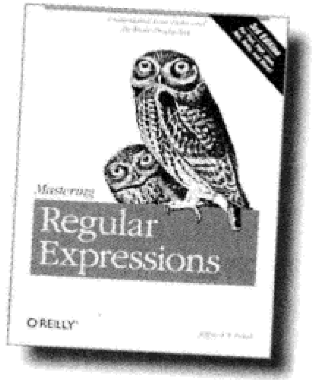
你的代码中可能永远不会用到这样一些语言特性，不过确实应该有所了解。请花些时间来了解何时使用以及在哪里使用这些语言特性。

请看本附录的**第10项**，那里列出了我最喜欢的Python书（除了这一本以外），它们都是不错的起点，从中可以更多地了解这些语言特性。



#5：正则表达式

处理文本数据时，Python相当自如。内置的字符串类型提供了很多方法，它们涵盖了大多数标准的字符串操作，如查找和分解字符串等。不过，如果需要从一个字符串中抽出某个特定部分，或者如果需要根据一个特定的规范在一个字符串中完成查找和替换，该怎么做呢？当然可以使用内置的字符串方法来实现这些问题的解决方案，不过（大多数人可能都会这样认为）使用**正则表达式**会更好一些。



如果你想了解更多有关内容，Jeff Friedl的这本正则表达式"圣经"很值得一读。也可以查看Python文档中的"re"模块。

考虑这样一个例子，要求你从phone_number字符串中抽出区号，这里使用了内置的字符串方法：

area_code.py - /Users/barryp/HeadFirstPython/Top Ten Things/area_code.py

```
phone_number = "Home: (555) 265-2901"

start = phone_number.find('(')

start = start+1
end = start+3

area_code = phone_number[start:end]

print('The area code is: ' + area_code)
```

查找开始"("。

计算区号在字符串的哪个位置。

抽出区号。

Ln: 12 Col: 0

这个代码可以很好地工作，不过如果为phone_number提供下面这个值，它就会出问题：

```
phone_number = "Cell (mobile): (555)-918-8271"
```

为什么这个电话号码会导致程序失败？试着运行这个代码，看看会发生什么……

使用**正则表达式**时，可以准确地指定你在找什么，相应地提高代码的健壮性：

area_code_re.py - /Users/barryp/HeadFirstPython/Top Ten Things/area_code_re.py

```
import re

phone_number = "Home: (555) 265-2901"

results = re.search('\((\d{3})\)', phone_number)
area_code = results.group(1)

print('The area code is: ' + area_code)
```

Ln: 10 Col: 0

这看起来有点奇怪，不过这个正则表达式在查找一个开始"("，后面是3个数字，然后是一个结束")"。这个规范往往用来查找区号，它不会像这个程序的另外那个版本那样很快崩溃。

#6: 关于Web框架

构建Web应用时，CGI是可行的，但是它有些“老土”。我们在第10章看到，Google的App Engine技术支持CGI，另外还支持WSGI以及很多Web框架技术。如果你没有部署到云，更愿意自力更生，你将有很多选择。下面是一些有代表性的例子。我的建议是：多尝试几个，看看哪一个最适合你。

用你喜欢的搜索引擎搜索以下关键词：Django、Zope、TurboGears、Web2py和Pylons。

“老前辈”……不过不要以为成熟即老套，这些绝对是一流的Web框架。

“新生代”：更简洁、更有前途，包含丰富的特性。

#7: 对象关系映射工具和NoSQL

在Python中处理基于SQL的数据库得到了充分的支持，而且标准库中包含了SQLite。当然，前提是你对SQL很熟悉，而且很乐于使用SQL来处理数据。

但是如果你不熟悉SQL呢？如果你很讨厌SQL呢？

对象关系映射工具（object relational mapper，ORM）是一种软件技术，允许你使用底层的一个基于SQL的数据库而不必了解SQL。并非是基于Python数据库API提供一个过程式接口，ORM通过方法调用和属性查找（而不是列和行）为数据提供了一个面向对象的接口。

很多程序员发现ORM对于处理所存储的数据集来说是一种更为自然的机制，Python社区已经创建并支持很多ORM。

其中最有意思的是SQLAlchemy，这个工具很流行，附录第6项中讨论的很多Web框架技术中都包含这个工具。除了非常流行外，SQL Alchemy有意思还有一个原因，因为它同时支持Python 2和Python 3，这让这个技术独树一帜（起码对现在来说）。

如果你发现自己越来越害怕SQL，就可以求助于ORM。当然，你已经用过一个类似的技术：Google App Engine的datastore API从风格上非常类似于主流Python ORM提供的API。

还有NoSQL

除了可以利用一些数据库技术来避免处理底层基于SQL的数据库，还出现了一类新兴的技术，利用这些技术你可以完全抛开SQL数据库。这些技术统称为NoSQL，这些数据工具为数据提供了一个候选的非SQL API，根本不使用基于SQL的数据库管理系统。由于这些技术相当新，围绕Python 2的工作比Python 3的相关工作更多，不过都很有必要了解。CouchDB和MongoDB与健壮的Python实现关联最紧密。如果你喜欢在Python字典中处理数据，希望采用数据库技术同样的方式存储数据，就需要考虑NoSQL，在这方面它实在太适合了。

#8:GUI编程

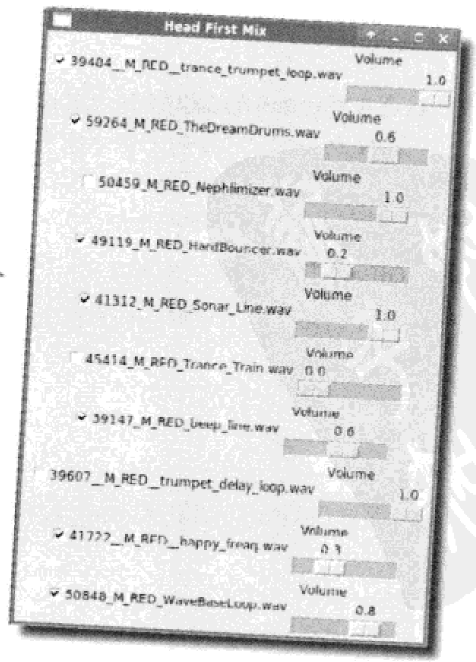
在这本书中，你已经创建过基于文本的界面、基于Web的界面以及在Android设备上运行的界面。不过，如果你想创建一个桌面应用，在你或用户的桌面上运行，该怎么做呢？你是不是毫无头绪，或者Python能提供帮助吗？

嗯……世事总是如此，Python预安装了一个GUI构建工具包，名为tkinter（Tk Interface的简写）。可以用tkinter创建一个可用，而且非常有用的图形化用户界面(GUI)，并部署到Mac OS X、Windows和Linux。利用最新版本的Tk，你开发的应用可以具备底层操作系统的特性，所以在Windows上运行时，你的应用就像是一个Windows桌面应用，如果在Linux上运行，则像是一个Linux桌面应用，诸如此类。

Python和tkinter代码只需写一次，就可以在任何地方运行，而且都能正常工作。要学习如何使用tkinter编程，有大量非常好的资源，其中最好的资源之一就是《Head First Programming》一书的最后几章，不过不好意思，这里有做广告的嫌疑，所以后面我不会再提到这本书了。

还存在另外一些GUI构建技术，最常谈到的是PyGTK、PyKDE、wxPython和PyQT工具包。但要注意，其中大多数工具包都面向Python 2，不过对Python 3的支持正在开发中。可以在网上搜索这里提到的项目名字来了解更多有关信息。

请看：这是"Head First Programming"创建的GUI之一……没错，之前说过我不会再提到那本书了，不过我实在忍不住想说这个GUI是不是很漂亮？:-)



#9：要避免的问题

说到使用Python时要避免的问题，其实并不多。最近Twitter上有这样的微博：





Python中确实存在线程，但是要尽可能避免。

这与Python的线程库没有任何关系，完全是因为Python的实现，特别是名为CPython的实现（你现在运行的可能正是这个实现）。Python使用一种称为**全局解释器锁**（Global Interpreter Lock，GIL）的技术来实现。它强制实行这样一个限制，要求Python只能在一个解释器进程中运行，即使有多个处理器可用。

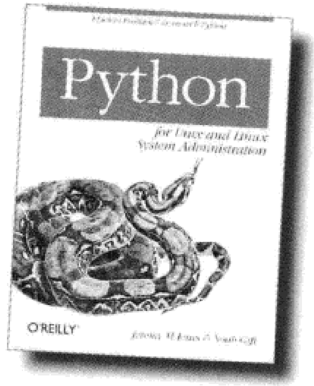
对于你来说，这意味着，如果你的程序使用了线程，尽管它的设计和实现都很棒，但是即使有多个处理器这个程序也不会运行得更快，因为它根本无法使用多个处理器。你的线程应用会串行运行，而且在很多情况下，甚至比没有用线程开发同样功能时慢得多。

要点：除非去除GIL限制（如果真的能去除）……否则不要在Python编程中使用线程。

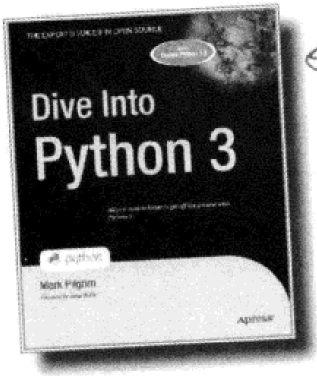
#10: 其他Python书

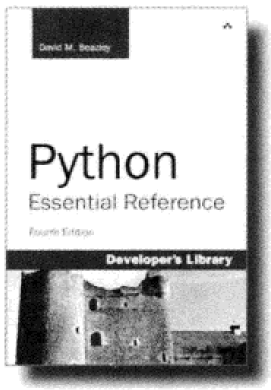
有很多全面介绍Python的好书，另外还有一些专门针对某个特定的问题领域。下面汇集了我最喜欢的Python书，在此向你强烈推荐。

如果你是一个系统管理员，这本Python书绝对适合你。

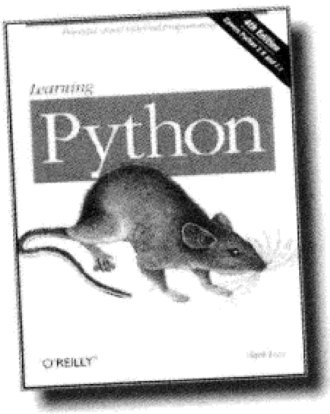


包括一个很不错的案例研究，介绍了如何将一个复杂的Python 2模块移植到Python 3。

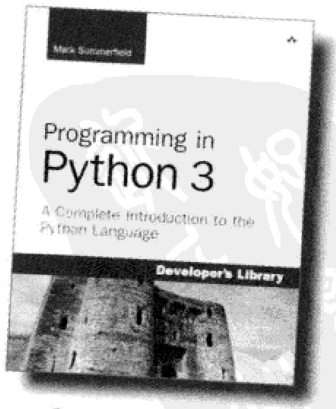




这是市场上最好的Python桌面开发参考书。



这本书有1200页，绝对是Python的一本权威语言参考，其中涵盖了Python的所有内容！



包括一些大型示例，其中用到一些重要技术：XML、解析以及其他高级语言特性。

索引

符号与数字

404 Error, from web server （404错误，来自Web服务器） 242
405 Error, from web server （405错误，来自Web服务器） 378
>>> (chevron, triple) IDLE prompt （>>>（三个大于号）IDLE 提示符） 4
: (colon) （:(冒号)）
in for loop (for循环中） 16
in function definition （函数定义中） 29
in if statement （if语句中） 20
, (comma) separating list items （,（逗号）分隔列表项） 7
{} (curly braces)（{}（大括号））
creating dictionaries （创建字典） 180
creating sets （创建集合） 166
= (equal sign)（=（等号））
assignment operator （赋值操作符） 7
in function argument definition （函数参数定义中） 63
(...) (parentheses)（(...) (括号)）
enclosing function arguments （包围函数参数） 29
enclosing immutable sets （包围不可变集合） 91
+ (plus sign) addition or concatenation operator （+（加号）加法或连接操作符） 138
(pound sign) preceding one-line comments （#（英镑符）引导单行注释） 38
@property decorator （@property修饰符） 250, 253, 285
? (question mark) parameter placeholder （?（问号）参数占位符） 321, 350
"..." or '...' (quotes) enclosing each list item （"..." 或 '...' (引号)包围各个列表项） 7
""" ...""" or ''' ...''' (quotes, triple) enclosing comments（""" ...""" 或 ''' ...''' (三重引号) 包围注释） 37
; (semicolon) separating statements on one line （; (分号) 分隔一行上的语句） 38
[...] (square brackets)（[...] (中括号)）
accessing dictionary items （访问字典项） 180, 212
accessing specific list items （访问特定列表项） 9, 18
enclosing all list items （包围所有列表项） 7, 18

A

"a" access mode （"a" 访问模式） 110
access modes for files （文件访问模式） 110, 133
addition operator (+)（加法操作符（+）） 138
Alt-N keystroke, IDLE （Alt-N按键, IDLE ） 5
Alt-P keystroke, IDLE （Alt-P按键, IDLE ） 5
AndFTP app （AndFTP应用） 288～289
Android apps（Android应用）
accepting input from（接收输入） 278～282, 295, 304～307
converting from Python code（由Python代码转换） 424～430
creating （创建） 274～277, 281～282
data for.*See* JSON data interchange format
integrating with SQLite（与SQLite集成） 342～348
running on phone（在手机上运行） 288～289
scripting layer for. *See* SL4A
troubleshooting （排错） 277
Android emulator（Android模拟器）
installing and configuring （安装和配置） 260～262
running scripts on（运行脚本） 264～265, 272～273, 283
Android Market （Android市场） 288
Android Virtual Device. *See* AVD
anonymous functions （匿名函数） 439
append() method, lists （append()方法，列表） 10, 14
apps. *See* Android apps; webapps
app.yaml file （app.yaml文件） 356, 395
arguments for functions（函数参数）
adding （增加） 52, 66～68
optional （可选） 63～64, 134
arrays（数组）
associative. *See* dictionaries
similarity to lists （与列表的相似性） 9～10, 17
as keyword （as关键字） 119, 138
assignment operator (=) （赋值操作符（=）） 7
associative arrays. *See* dictionaries
attributes, class （属性，类） 190, 194, 212

authorization, user （授权，用户） 389～393

AVD (Android Virtual Device) （AVD（Android虚拟设备）） 261, 291

B

“batteries included” （功能齐全） 32

BIFs. *See* built-in functions

BigTable technology （BigTable技术） 354, 359

blue text in IDLE （IDLE中的蓝色文本） 4

books（图书）
- Dive Into Python 3 (CreateSpace) 445
- Head First HTML with CSS & XHTML (O'Reilly) 374
- Head First Programming (O'Reilly) 443
- Head First SQL (O'Reilly) 313
- Learning Python (O'Reilly) 445
- Mastering Regular Expressions (O'Reilly) 440
- Programming in Python 3 (Addison-Wesley Professional) 445
- Python Essential Reference (Addison-Wesley Professional) 445
- Python for Unix and Linux System Administration (O'Reilly) 445

braces. *See* curly braces

brackets, regular.*See* parentheses

brackets, square. *See* square brackets

BSD, running CGI scripts on （BSD，运行CGI脚本） 239

build folder （build文件夹） 42

built-in functions (BIFs). *See* specific functions
- displayed as purple text in IDLE （IDLE中显示为紫色文本） 4
- help on （帮助） 21
- importing of, not needed （导入，不需要） 55
- namespace for （命名空间） 55
- number of, in Python （数目，Python中） 21
- __builtins__ namespace （__builtins__命名空间） 55, 71

C

cascading style sheet (CSS) （层叠样式表（CSS）） 374～375

case sensitivity of identifiers （标识符的大小写敏感性） 17

cgi-bin folder （cgi-bin文件夹） 234, 235

CGI (Common Gateway Interface) scripts （CGI（通用网关接口）脚本）） 217, 235, 243, 253. *See* WSGI
- location of （位置） 234, 235
- running （运行） 239
- running from Android （从Android运行） 264～265, 272～273, 283
- sending data to （发送数据） 300～303
- tracking module for （跟踪模块） 248～249
- troubleshooting （排错） 242, 247～250
- writing （写） 236～238, 244～246
- writing for Android. *See* SL4A

cgi library （cgi库） 300

cgitb module （cgitb模块） 248～249, 253

chaining（串链）
- functions （函数） 146, 172
- methods （方法） 142, 172

chevron, triple (>>>) IDLE prompt（三个大于号（>>>）IDLE提示符） 4

chmod command （chmod命令） 239, 253

classes（类） 189～191
- attributes of（属性） 190, 194, 212
- benefits of（好处） 189
- converting data to dictionary（数据转换为字典） 285～286
- defining （定义） 190～193, 194, 195～196
- inherited （继承） 204～209, 212
- instances of （实例） 190, 191, 194, 195～196
- metaclasses （元类） 439
- methods of （方法） 190, 195～196, 198～200
- in modules （模块中） 209, 212

class keyword （class关键字） 191, 212

close() method, database connection （close()方法，数据库连接） 315, 350

close() method, files （close()方法，文件） 75, 103

code editors 35, 436. *See* IDLE

colon (:)（冒号（:））
- in for loop (for循环中） 16
- in function definition （函数定义中） 29
- in if statement （if语句中） 20

comma (,) separating list items （逗号（,）分隔列表项） 7

comments （注释） 37～38

commit() method, database connection （commit()方法，数据库连接） 315, 350

Common Gateway Interface scripts. *See* CGI scripts

comprehension, list （推导，列表） 154～159, 172, 409～411, 432

concatenation operator (+)（连接操作符（+）） 138

conditional list comprehension （条件列表推导） 409～411, 432

conditions. *See* if/else statement

connection, database（连接，数据库）
  closing （关闭） 314, 315
  creating （创建） 314, 315

connect() method,sqlite3 （connect()方法，sqlite3） 315, 350

context management protocol（上下文管理协议） 120

Controller, in MVC pattern （控制器，MVC模式中） 221
  for GAE webapps （GAE应用） 359, 370～373
  for webapps （Web应用） 234～238, 244～246

"copied" sorting （"复制"排序） 144, 145～146, 172

CREATE TABLE statement, SQL （CREATE TABLE语句，SQL） 317, 319～320

CSS (cascading style sheet) （CSS（层叠样式表）） 374～375

CSV format, converting to Python data types （CSV格式，转换为Python数据类型） 401～405

curly braces ({})（大括号 ({}))
  creating dictionaries （创建字典） 180
  creating sets （创建集合） 166

cursor() method,database connection （cursor()方法，数据库连接） 315, 350

custom code （定制代码） 131

custom exceptions （定制异常） 439

D

data（数据）
  for Android apps. *See* JSON data interchange format
  bundling with code. *See* classes
  duplicates in, removing （重复，删除） 161～163, 166～167
  external. *See* database management system; files
  for GAE webapps. *See* datastore, for GAE
  nonuniform, cleaning （不统一，清理） 148～153
  race conditions with （竞态条件） 309～310
  Robustness Principle for （健壮性原则） 384～387
  searching for closest match （搜索最接近的匹配） 416～417
  sending to web server （发送到Web服务器） 275, 291
  sorting （排序） 144～147, 172
  storing. *See* persistence
  transforming, list comprehensions for （转换，列表推导） 154～159

database API （数据库API） 314～315, 350

database management system （数据库管理系统） 312
  closing connection to （关闭连接） 314, 315
  commit changes to data （提交数据修改） 314, 315, 350
  connecting to （连接到） 314, 315
  cursor for, manipulating data with （游标，处理数据） 314, 315
  designing （设计） 316～318
  inserting data into （插入数据） 321, 324, 348
  integrating with Android apps （集成Android应用） 342～348
  integrating with webapps （集成Web应用） 327～341
  managing and viewing data in （管理和查看数据） 326
  process for interacting with （交互过程） 314～315
  querying （查询） 322, 332～333
  rollback changes to data （回滚数据修改） 314, 315, 350
  schema for （模式） 317
  SQLite for. *See* SQLite
  tables in （表） 317, 319～320, 350

data folder （data文件夹） 234

data interchange format. *See* JSON data interchange format

data objects. *See* specific data objects
  getting next item from （得到下一项） 54
  ID for （ID） 54
  length of, determining （长度，确定） 32
  names of. *See* identifiers

datastore, for GAE （datastore，GAE） 359～360, 380～383, 384～387, 395

data types（数据类型）
  converting CSV data into （CSV数据转换为） 401～405
  converting strings to integers （字符串转换为整数） 54
  in datastore （datastore中） 360
  immutable （不可变） 91, 103, 116, 138
  for JSON （JSON) 285
  for list items (列表项） 8, 12

date and time data（日期和时间数据）
  format compatibility issues （格式兼容性问题） 418～423
  property type for （属性类型） 362, 384～385

db.Blob() type （db.Blob()类型） 360

db.DateProperty() type （db.DateProperty()类型） 360

db.IntegerProperty() type （db.IntegerProperty()类型） 360

db.StringProperty() type （db.StringProperty()类型） 360, 385

db.TimeProperty() type （db.TimeProperty()类型） 360

db.UserProperty() type （db.UserProperty()类型） 360, 390

decision statement. *See* if/else statement

decorators, function （修饰符，函数） 439

def keyword （def关键字） 29, 191, 212

dialogCreateAlert() method,Android API （dialogCreateAlert()方法,Android API ） 274, 276, 280

dialogGetInput() method,Android API （dialogGetInput()方法,Android API ） 295, 304～306

dialogGetResponse() method,Android API （dialogGetResponse()方法,Android API ） 274, 276, 278, 280

dialogGetSelectedItems() method,Android API （dialogGetSelectedItems()方法,Android API ） 278, 280

dialogSetItems() method,Android API （dialogSetItems()方法,Android API ） 279, 280

dialogSetNegativeButtonText() method,Android API （dialogSetNegativeButtonText()方法,Android API ） 274, 276

dialogSetPositiveButtonText() method,Android API （dialogSetPositiveButtonText()方法,Android API ） 274, 276, 280

dialogSetSingleChoiceItems() method,Android API （dialogSetSingleChoiceItems()方法,Android API ） 274, 276

dialogShow() method,Android API （dialogShow()方法,Android API ） 276, 280

dict() factory function （dict()工厂函数） 180, 212

dictionaries（字典） 178～182, 212
- accessing items in（访问项） 180, 212
- compared to lists（与列表比较） 179
- converting class data to （转换类数据为） 285～286
- creating （创建） 180, 182, 186
- dictionaries within （内的字典） 407～409
- keys for （键） 178, 180, 212
- populating （填充） 180, 212
- reading CSV data into （读入CSV数据） 403～404
- values of （值） 178, 180, 212

dir() command （dir()命令） 225

directory structure. *See* folder structure

dist folder （dist文件夹） 42

distribution（发布）
- creating （创建） 40～42
- updating （更新） 60～61, 65
- uploading to PyPI （上传到PyPI ） 48

Dive Into Python 3 (CreateSpace) 445

djangoforms.ModelForm class （djangoforms.ModelForm类） 368

Django Project（Django项目）
- Form Validation Framework （表单验证框架） 368～369, 395
- templates （模板） 363～366, 395

doctest framework （doctest框架） 438

documentation for Python 3 （Python 3文档） 3, 80, 103

dot notation （点记法） 10, 194, 196

double quotes. *See* quotes

dump() function, pickle （dump()函数, pickle） 133～134, 138

dumps() function, json （dumps()函数json） 269, 272, 281, 291

dynamic content （动态内容） 216, 217

E

Eclipse editor （Eclipse编辑器） 35

editors 35, 436. *See* IDLE

elif keyword （关键字） 108. *See* if/else statement

else keyword. *See* if/else statement

emacs editor （emacs编辑器） 35, 436

enable() function, cgitb （enable()函数， cgitb） 248, 253

end_form() function, yate （end_form()函数, yate） 231, 233

entities, in datastore （实体， datastore中） 360, 395

enumerate() built-in function （enumerate()内置函数） 54

environ dictionary （environ字典） 300, 350

equal sign (=)（等号（=））
- assignment operator （赋值操作符） 7
- in function argument definition （函数参数定义中） 63

errors. *See* exception handling; troubleshooting

exception handling 88～95, 103. *See* troubleshooting
- benefits of（好处） 95, 100
- closing files after （关闭文件） 114～115, 120～123
- custom exceptions （定制异常） 439
- defining with try/except statement （用try/except语句定义） 89, 91～94
- ignoring found errors （忽略找到的错误） 93～94
- IndexError exception （IndexError异常） 17
- IOError exception （IOError异常） 103, 112～114, 117～119
- for missing files （缺少文件） 96～98
- NameError exception （NameError异常） 44, 118
- PickleError exception （PickleError异常） 133～134
- specific errors, checking for （特定错误， 检查） 101～102
- specific errors, details about （特定错误， 详细信息） 117～119
- TypeError exception （TypeError异常） 56～57, 116, 247～249, 283～285
- ValueError exception （ValueError异常） 78～79, 81～82, 103

exception objects （异常对象） 119, 138

except keyword. *See* try/except statement

execute() method,cursor （execute()方法， 游标） 315, 322, 324, 350

extend() method,lists （extend()方法， 列表） 10

F

F5 key, IDLE （F5按键，IDLE） 39, 44, 49, 71
factory functions （工厂函数） 166
favicon.ico file, for webapp （favicon.ico文件，web应用） 234
fetchall() method,cursor （fetchall()方法，游标） 322
fetchmany() method,cursor （fetchmany()方法，游标） 322
fetchone() method,cursor （fetchone()方法，游标） 322
FieldStorage() method,cgi（FieldStorage()方法，cgi） 244, 253, 296, 300, 350
files. *See* persistence
  access modes for （访问模式） 110, 133
  appending data to （追加数据） 110
  checking for existence of （检查是否存在） 118
  closing （关闭） 75, 110
  closing after exception （异常后关闭） 114～115, 120～123
  CSV format, converting to Python data types（CSV格式，转换为Python数据） 401～405
  exceptions involving, determining type of（异常，确定类型） 117～119
  flushing（刷新输出） 110
  missing, exception handling for （缺少，异常处理） 96～98
  opening （打开） 75, 109～110
  opening in binary access mode （以二进制访问模式打开） 133
  reading data from （读取数据） 75～78, 142～143
  rewinding（退回） 76
  splitting lines in （分解行） 77～78
  writing （写） 110～113
  writing, custom formats for （写，定制格式） 126～130
  writing, default format for （写，默认格式） 124～125
  writing, pickle library for. *See* pickle library
finally keyword （finally关键字） 115, 138
find() method,strings （find()方法，字符串） 84～86, 103
Firefox, SQLite Manager for 326
folder structure（文件夹结构）
  for distribution （发布） 42
  for GAE （GAE） 356, 370
  for webapps （Web应用） 234
for loop （for循环） 15～17, 32
  compared to list comprehension （与列表推导比较） 432
  nesting （嵌套） 19～22
forms, HTML （表单，HTML） 295
  creating from template （由模板创建） 296～299
  Form Validation Framework for （表单验证框架） 368～369
  input restrictions for （输入限制） 376～377, 384～387
  sending data to CGI scripts （向CGI脚本发送数据） 300～303
  stylesheets for （样式表） 374～375
Form Validation framework （表单验证框架） 368～369, 395
405 Error, from Web Server （405错误，来自Web服务器） 378
404 Error, from Web Server （404错误，来自Web服务器） 242
Friedl, Jeff (author, Mastering Regular Expressions) （Friedl, Jeff，Mastering Regular Expressions的作者） 440
from statement （from语句） 46, 49
functional programming concepts （函数式编程概念） 157
function decorators （函数修饰符） 439
functions（函数）
  adding arguments to （增加参数） 52, 66～68
  anonymous （匿名） 439
  built-in. *See* built-in functions (BIFs)
  chaining （串链） 146, 172
  creating （创建） 28～30, 170～171
  optional arguments for （可选参数） 63～64, 134
  recursive （递归） 31
  sharing. *See* modules

G

GAE (Google App Engine) 354
  configuration and setup for（配置和建立） 356～357
  controller code for（控制器代码） 370～373
  data modeling with（数据建模） 360～362
  datastore for（datastore） 359, 380～383, 384～387, 395
  deploying webapps to Google cloud （Web应用部署到Google云） 391
  folder structure for（文件夹结构） 356, 370
  form generation for（表单生成） 368～369
  form input restrictions for（表单输入限制） 376～377, 384～387
  form stylesheets for （表单样式表） 374～375
  MVC pattern used by （使用MVC模式） 359
  SDK for, downloading（SDK，下载） 355
  troubleshooting （排错） 378
  user authorization for（用户授权） 389～393
  view for, desigining （视图，设计） 363～369
GAE Launcher 357, 391, 395
garbage collection（垃圾回收） 116
generators （生成器） 439

get() method,GAE （()方法，GAE） 370, 379, 395
GET web request （GET web请求） 370
GIL (Global Interpreter Lock)（GIL（全局解释器锁） 444
glob module （glob模块） 237, 253
Google App Engine. *See* GAE
Google BigTable technology （Google BigTable技术） 354, 359
GQL (Google Query Language) API（GQL（Google查询语言）API） 359
green text in IDLE （IDLE中的绿色文本） 4
GUI (graphical user interface), building （GUI（图形用户界面），构建） 443

H

hashes. *See* dictionaries
header() function, yate （header()函数, yate） 231, 233
Head First HTML with CSS & XHTML (O'Reilly) 374
Head First Programming (O'Reilly) 443
Head First SQL (O'Reilly) 313
help() built-in function （help()内置函数） 80, 103
HTML
  generating for webapp interface （为Web应用界面生成） 230～231
  learning （学习） 226
  templates for, with Django （模板，Django） 363～366
HTML forms. *See* forms, HTML
HTTP server （HTTP服务器） 235
http.server module （http.server模块） 235, 253

I

id() built-in function （id()内置函数） 54
IDE 436. *See* IDLE
identifiers （标识符） 7, 17, 32
IDLE 3～5, 32
  colored syntax used in （所用区分颜色的语法） 4
  indenting enforced in （强制缩进） 4
  preferences, setting（首选项，设置） 5
  prompt in (>>>) （提示符） 4
  recalling and editing code statements （回退和编辑代码语句） 5
  running or loading code in （运行或加载代码） 39, 44, 49, 71
  TAB completion （TAB完成） 5
if/else statement （if/else语句） 20, 32
  elif keyword （elif关键字） 108
  in list comprehension（列表推导中） 432
  negating conditions in （取反条件） 86, 103
images folder （images文件夹） 234
immutable data types （不可变数据类型） 138
  lists （列表） 91, 103, 116
  numbers （数值） 116
  strings （字符串） 116
import statement （import语句） 43, 46, 49, 71
include_footer() function, yate （include_footer()函数, yate） 230, 232, 233
include_header() function, yate （include_header()函数, yate） 230, 232
indentation rules（缩进规则）
  enforced in IDLE （IDLE中强制） 4
  for for loops （for循环） 16
  for function definitions （函数定义） 29
  for if statement （if语句） 20
IndexError exception （IndexError异常） 17
index.html file, for webapp （index.html文件，Web应用） 234
inherited classes （继承类） 204～209, 212
__init__() method（__init__()方法） 191, 212
in operator （in操作符） 16, 118, 138
"in-place" sorting （"原地"排序） 144, 145, 172
input（输入）
  from Android apps （Android应用） 278～282, 295, 304～307
  HTML forms for. *See* forms, HTML
  from keyboard after screen prompt （屏幕提示后的键盘输入） 413～414, 432
input() built-in function （input()内置函数） 413～414, 432
insert() method,lists （insert()方法，列表） 10, 14
INSERT statement, SQL （INSERT语句，SQL） 321, 324, 348
instances of classes （类实例） 190, 191, 194, 195～196
int() built-in function （int()内置函数） 54
integers, converting strings to （整数，字符串转换为整数） 54
interface. *See* View, in MVC pattern
IOError exception （IOError异常） 103, 112～114, 117～119

I/O (input/output), handling. *See* files
isinstance() built-in function （isinstance()内置函数） 20～22, 32
iterations（迭代）
for loop （for循环） 15～17, 19～22, 32
generating with range() function （用range()函数生成） 54～56
while loop (while循环) 16～17

J

JSON data interchange format（JSON数据交换格式） 266～267, 291
API for, using （API，使用） 269～272
browser differences with （浏览器差别） 272
data types supported by （支持的数据类型） 285
incompatibility with pickle data objects（与pickle数据对象不兼容） 284～285

K

KDevelop IDE 436
keys, in dictionary （键，字典中） 178, 180, 212
keywords, displayed as orange text in IDLE （关键字，IDLE中显示为橙色） 4

L

lambda expression （λ表达式） 439
Learning Python (O'Reilly) 445
len() built-in function （len()内置函数） 10, 32
lib folder （lib文件夹） 42
Linux
code editors for （代码编辑器） 35
GAE log messages on （GAE日志消息） 378
IDEs for （IDE） 436
installing Python 3 on （安装Python 3） 3
running CGI scripts on （运行CGI脚本） 239, 272
running GAE Launcher on （运行GAE Launcher） 357
transferring files to Android device（文件传输到Android设备） 288
list() built-in function （list()内置函数） 54
list comprehension （列表推导） 154～159, 172, 409～411, 432
lists 32. *See* data objects
adding items to （增加项） 10～14
bounds checking for （越界检查） 17
classes inherited from （由列表继承的类） 204～208
compared to dictionaries （与字典比较） 179
compared to sets （与集合比较） 167
creating （创建） 6～7, 54
data types in （数据类型） 8, 12
duplicates in, removing （重复，删除） 161～163
extracting specific item from （从中抽取特定项） 175～176
getting next item from （得到下一项） 54
identifiers for （标识符） 7
immutable （不可变） 91, 103, 116
iterating （迭代） 15～17, 157
length of, determining （长度，确定） 10, 32
methods for （方法） 10
nested, checking for （嵌套，检查） 20～22
nested, creating （嵌套，创建） 18～19
nested, handling （嵌套，处理） 23～25, 28～31
numbered, creating （编号，创建） 54
reading CSV data into （读入CSV数据） 403～404
removing items from （删除项） 10
similarity to arrays （与数组的相似性） 9～10, 17
slice of （分片） 160, 172
load() function, pickle （load()函数, pickle） 133, 138
loads() function, json （loads()函数，json） 269, 276, 280, 291
locals() built-in function （locals()内置函数） 118, 138
loops. *See* iterations

M

Mac OS X
code editors for （代码编辑器） 35
GAE log messages on （GAE日志消息） 378
IDEs for （IDE） 436
installing Python 3 on （安装Python 3） 3
running CGI scripts on （运行CGI脚本） 239, 272
running GAE Launcher on （运行GAE Launcher） 357
transferring files to Android device（文件传输到Android设备） 288
__main__ namespace （__main__命名空间） 45
MANIFEST file （MANIFEST文件） 42
mappings. *See* dictionaries

Mastering Regular Expressions (O'Reilly) 440
metaclasses （元类） 439
methods （方法） 190. *See* specific methods
  chaining （串链） 142, 172
  for classes （类） 195～196, 198～200
  creating （创建） 212
  results of, as attributes （结果，作为属性） 250, 253
  self argument of （self参数） 212
ModelForm class, djangoforms （ModelForm类，djangoforms） 368
Model, in MVC pattern （模型，MVC模式中） 221
  for GAE webapps （GAE应用） 359, 360～362
  for webapps （Web应用） 222～225
Model-View-Controller pattern. *See* MVC pattern
modules （模块） 34～36, 71
  adding functionality to （增加功能） 50～52
  classes in （类） 209, 212
  creating （创建） 35
  distribution for, creating （发布，创建） 40～42
  distribution for, updating （发布，更新） 60～61, 65
  distribution for, uploading to PyPI （发布，上传到PyPI） 48
  importing （导入） 43～44, 46
  loading in IDLE （加载到IDLE中） 39, 49, 71
  locations for （位置） 38, 49
  namespaces for （命名空间） 45～46, 71
  in Python Standard library （Python标准库中） 36
  third-party （第三方） 36
Monty Python 17
multiple inheritance （多重继承） 209
MVC (Model-View-Controller) pattern （MVC（模型－视图－控制器）模式） 221, 232, 253, 359
  Controller （控制器） 234～238, 244～246, 370～373
  Model （模型） 222～225, 360～362
  View （视图） 226～233, 363～369

N

NameError exception （NameError异常） 44, 118
names. *See* identifiers
namespaces （命名空间） 45～46, 71
next() built-in function （next()内置函数） 54
NoSQL 359, 442
NotePad editor （NotePad编辑器） 35
not in operator （not in操作符） 161～162
not keyword （not关键字） 86, 103
numbered lists （编号列表） 54

O

object relational mapper. *See* ORM (object relational mapper)
objects. *See* data objects
open() built-in function （open()内置函数） 75, 103, 109～110
orange text in IDLE （IDLE中的橙色文本） 4
ORM (object relational mapper) （ORM（对象关系映射工具）） 442
os module （os模块） 76, 300, 350

P

para() function, yate （para()函数, yate） 231, 233
parentheses ((...)) （括号（(...)））
  enclosing function arguments （包围函数参数） 29
  enclosing immutable lists （包围不可变列表） 91
pass statement （pass语句） 93, 103
persistence （持久存储） 105
  pickle library for （pickle库） 132～137
  reading data from files （从文件读取数据） 222～224
  writing data to files （数据写至文件） 110～113, 222～224
PickleError exception （PickleError异常） 133～134
pickle library （pickle库） 132～137, 138
  data modeling using （数据建模） 222～224
  incompatibility with JSON data types （与JSON数据类型的兼容性） 284～285
  transferring data to a database （数据传输到数据库） 321～325
plus sign (+) addition or concatenation operator （加号（+）加法或连接操作符） 138
pop() method, lists （pop()方法，列表） 10, 175～176
post() method, GAE （post()方法，GAE） 379～383, 395
POST web request （POST Web请求） 379
pound sign (#) preceding one-line comments （英镑符（#）引导单行注释） 38
print() built-in function （print()内置函数） 10, 32, 124～125
  disabling automatic new-line for （禁用自动换行） 56, 71
  displaying TAB character with （显示TAB字符） 56
  writing to a file （写至文件） 110, 128, 138
Programming in Python 3 (Addison-Wesley Professional) 445
properties, in datastore （属性，datastore中） 360, 395

@property decorator （@property修饰符） 250, 253, 285
purple text in IDLE （IDLE中的紫色文本） 4
put() method,GAE （put()方法，GAE） 395
.pyc file extension （.pyc文件扩展名） 42, 49
.py file extension （.py文件扩展名） 35
PyPI (Python Package Index) 36
registering on website（网站上注册） 47
uploading distributions to （上传发布包） 48
uploading modules to （上传模块） 209
Python 2
compared to Python 3 （与Python 3比较） 17
raw_input() built-in function （raw_input()内置函数） 432
running on Android smartphones（在Android智能手机上运行） 258～259, 291
using with Google App Engine （用于Google App Engine） 355
Python 3
compared to Python 2 （与 Python 2比较） 17
documentation for （文档） 3, 80, 103
editors for （编辑器） 35, 436
installing （安装） 3
interpreter for. *See* IDLE
learning （学习） 445
python3 command（python3命令）
building a distribution（构建一个发布） 41
checking for Python version （检查Python版本） 3
installing a distribution （安装一个发布） 41
uploading a new distribution （上传一个新发布） 68
Python Essential Reference (Addison-Wesley Professional) 445
Python for Unix and Linux System Administration (O'Reilly) 445
Python, Monty 17
Python Package Index. *See* PyPI
Python Standard library （Python标准库） 36

Q

querying a database （查询数据库） 322, 332–333
question mark (?) parameter placeholder （问号（?）参数占位符） 321, 350
quotes ("..." or '...') enclosing each list item （引号（"..." 或 '...' ）包围各列表项） 7
quotes, triple (""" ...""" or ''' ...''') enclosing comments （引号，三重（""" ...""" 或 ''' ...''' ）包围注释） 37

R

"r" access mode （"r" 访问模式） 110
race conditions（竞态条件） 309～310
radio_button() function, yate （radio_button()函数, yate） 231, 233
range() built-in function （range()内置函数） 54～56, 71
raw_input() built-in function （raw_input()内置函数） 432
readline() method,files （readline()方法，文件） 76, 103, 142
recursion （递归） 31
regular brackets. *See* parentheses
regular expressions（正则表达式） 440
re module （re模块） 440
remove() method,lists （remove()方法，列表） 10
render() function, template （render()函数，模板） 364, 366
Robustness Principle （健壮性原则） 384～387
rollback() method,database connection （rollback()方法，数据库连接） 315, 350
runtime errors 88. *See* exception handling; troubleshooting

S

schema, database （模式，数据库） 317
scoping of variables （变量作用域） 437
Scripting Layer for Android. *See* SL4A
scripts. *See* CGI scripts; SL4A
sdist command （sdist命令） 41
seek() method, files （seek()方法，文件） 76, 103
SELECT/OPTION, HTML tag （SELECT/OPTION，HTML标记） 376
SELECT statement, SQL （SELECT语句，SQL） 322, 332～333
self argument（self参数） 192～193, 212
self.request object （self.request对象） 379, 395
self.response object（self.response对象） 372, 379, 395
semicolon (;) separating statements on one line（分号（;）分隔一行上的语句） 38
set() built-in function （set()内置函数） 166, 172
sets （集合） 166, 167, 172
setup() built-in function （setup()内置函数） 40
setup.py file （setup.py文件） 40, 71
single quotes. *See* quotes
SL4A (Scripting Layer for Android) 258, 291

adding Python to （增加Python） 263
Android apps, creating （Android应用，创建） 274～277
automatic rotation mode, setting （自动旋转模式，设置） 264
documentation for （文档） 274
installing （安装） 262
Python versions supported（支持的Python版本） 258～259
slice of a list（列表分片） 160, 172
smartphones, apps on. *See* Android apps
sorted() built-in function （sorted()内置函数） 144～147, 153, 158, 172
sort() method,lists （sort()方法，列表）144～145, 153, 172
split() method,strings （split()方法，字符串） 77～78, 80～81, 103, 142
SQL Alchemy 442
SQLite 313, 350
closing connection to （关闭连接） 314, 315
committing data to （提交数据） 314, 315
connecting to （连接到） 314, 315
cursor for, manipulating data with （游标，处理数据） 314, 315
designing database （设计数据库） 316～318
inserting data into （插入数据） 321, 324, 348
integrating with Android apps （集成Android应用） 342～348
integrating with webapps （集成Web应用） 327～341
managing data in （管理数据） 326
process for interacting with （交互过程） 314～315
querying （查询） 322, 332～333
rollback changes to data （回滚数据修改）314
schema for database （数据库模式） 317
tables in, creating （表，创建） 319～320
sqlite3 command （sqlite3命令） 326
sqlite3 library （sqlite3库） 313, 315, 350
SQLite Manager, for Firefox 326
SQL (Structured Query Language) 313, 350. 参见 NoSQL; SQLite; ORM
square brackets ([...])（中括号（[...]））
accessing dictionary items （访问字典项） 180, 212
accessing specific list items （访问特定列表项） 9, 18
enclosing all list items （包围所有列表项） 7, 18
standard error (sys.stderr) （标准错误（sys.stderr）） 248, 291
standard input (sys.stdin) （标准输入（sys.stdin）） 291
Standard Library, Python （标准库，Python） 36
standard output (sys.stdout) （标准输出 （sys.stdout））126～128, 291
start_form() function, yate （start_form()函数, yate） 231, 233
start_response() function, yate （start_response()函数, yate） 230, 232
static content （静态内容） 216, 217
static folder （static文件夹） 370
str() built-in function （str()内置函数） 119, 138
strings（字符串）
concatenating （连接） 138
converting other objects to （转换其他对象） 119
converting to integers （转换为整数） 54
displayed as green text in IDLE （IDLE中显示为绿色文本） 4
finding substrings in （查找子串） 84～86
immutable （不可变） 116
sorting （排序） 148
splitting （分解） 77～78, 80～81
substitution templates for （替换模板） 230, 253
strip() method,strings （strip()方法，字符串） 108, 138, 142
Structured Query Language. *See* SQL
stylesheets for HTML forms（HTML表单的样式表） 374～375
suite（组）16, 29, 32
sys module （sys模块） 291
sys.stdout file （sys.stdout文件） 126～128, 138

T

TAB character, printing（TAB字符，打印） 56
TAB completion, IDLE（TAB完成，IDLE） 5
tables, database（表，数据库） 317, 319～320, 350
target identifiers, from split() method（目标标识符，split()方法） 77, 91
.tar.gz file extension （.tar.gz文件扩展名） 42
Template class （Template类） 230, 253
template module（template模块） 364
templates folder （template文件夹） 234, 370
templates for GAE （GAE模板） 363～366, 395
testing code （测试代码） 438
TextMate editor（TextMate编辑器） 35, 436
third-party modules（第三方模块） 36
threads（线程） 444
time data（时间数据）
format compatibility issues（格式兼容性问题） 418～423
property type for（属性类型） 362, 384～385
time module（time模块） 419, 432
Tk Interface (tkinter) 443

traceback 88, 103. *See* exception handling; troubleshooting
tracing code （跟踪代码0）58～59
triple chevron (>>>) IDLE prompt （三个大于号（(>>>)）IDLE提示符） 4
triple quotes (“” “...” ” ” or ‘’ ‘...’ ’ ’) enclosing comments （三重引号（ “” “...” ” ” 或 ‘’ ‘...’ ’ ’ ）包围注释） 37
troubleshooting. *See* exception handling
404 Error, from web server （404错误，来自Web服务器） 242
405 Error, from web server （405错误，来自Web服务器） 378
Android apps （Android应用） 277
GAE webapps （Web应用） 378
testing code（测试代码） 438
tracing code （跟踪代码） 58～59
try/except statement （try/except语句） 89, 93～94, 101～102, 103
finally keyword for （finally关键字） 115, 138
with statement and （with语句） 120～123
tuples （元组） 91, 103, 116
TypeError exception （TypeError异常） 56～57, 116, 247～249, 283～285

U

u_list() function, yate（u_list()函数, yate） 231, 233
unittest module（unittest模块） 438
urlencode() function, urllib （urlencode()函数， urllib ） 291
urllib2 module（urllib2模块） 291
urllib module（urllib模块） 291
urlopen()function,urllib2（urlopen()函数, urllib2 ） 291
user authorization（用户授权） 389～393
user input. *See* forms, HTML; input
UserProperty() type, db（UserProperty()类型， db） 390

V

ValueError exception（ValueError异常） 78～79, 81～82, 103
values, part of dictionary（值，字典的一部分） 178, 180, 212
variables, scope of（变量， 作用域） 437
vi editor（vi编辑器） 35, 436
View, in MVC pattern （视图，MVC模式中） 221
for GAE webapps （GAE应用） 359, 363～369
for webapps （web应用） 226～233
vim editor （vim编辑器） 436

W

“w” access mode（“w”访问模式） 110
“w+” access mode（“w+”访问模式） 110
“wb” access mode（“wb”访问模式） 133
webapps（web应用） 215～217, 253
controlling code for（控制代码） 221, 234～238, 244~246
data modeling for（数据建模） 221, 222～225
designing with MVC（用MVC设计） 221
design requirements for（设计需求） 218～220
directory structure for（目标结构） 234
Google App Engine for. *See* GAE
input data, sending to CGI scripts（输入数据，发送到CGI脚本） 300～303
input forms for. *See* forms, HTML
SQLite used with （使用SQLite） 327～341
view for（视图） 221, 226～233
Web-based applications. *See* webapps
Web frameworks 441. *See* CGI; WSGI
Web request（Web请求） 216, 253, 395
Web response（Web响应） 216～217, 253, 395
Web Server（Web服务器） 216～217, 235
Web Server Gateway Interface (WSGI) 356, 370. *See* CGI scripts
While loop（while循环） 16～17, 55
WingIDE editor（WingIDE编辑器） 35
WingWare Python IDE 436
With statement（with语句） 120～123, 138
WSGI (Web Server Gateway Interface) 356, 370. *See* CGI scripts

Y

yate (Yet Another Template Engine) library（yate库） 226～233
yield expression（yield表达式）439